Universidad
Carlos III de Madrid
www.uc3m.es

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Escuela Politécnica Superior

"Introducción a la plataforma Arduino y al Sensor ultrasónico HC-SR04"

Experimentado en una aplicación para medición de distancias

Autor: Virginia Martínez Fuentes

Tutor: Antonio Berlanga de Jesús

Miércoles, 24 de septiembre de 2014

A mis profesores, compañeros y amigos. Por su dedicación y por la confianza que han depositado en mí a lo largo de estos años.

Y en especial, a mi hermana.

Índice de Contenidos

Índice de ilustraciones

Índice de gráficas

Prueba 6. Distancia a la mano de una persona

Introducción

Contexto actual

Estado del arte

Con la creciente evolución de las TIC (Tecnologías de la Información y la Comunicación) en los últimos años, somos cada vez más, las personas que buscamos **adaptar las diversas tecnologías y dispositivos** que tenemos a nuestro alcance, a nuestras actividades y necesidades cotidianas. A diario, utilizamos ordenadores y dispositivos móviles: en casa, en el estudio, en el trabajo, en las relaciones personales y en la práctica de actividades lúdicas. Elegir la tecnología adecuada según las necesidades y propósitos que busquemos satisfacer, será una decisión importante que nos llevará o no, a conseguir nuestros objetivos.

En Internet se puede encontrar gran cantidad de información, pero es necesario contrastarla. Para desarrollar cualquier trabajo de investigación es, por tanto, fundamental **consultar diversas fuentes de información** (y así ha sido en este caso, a través de multitud de manuales, sitios web y foros) antes de establecer las conclusiones que nos lleven a la elección de una tecnología u otra.

En torno a la evolución de la tecnología surgen diferentes **comunidades o grupos de usuarios** que quieren conocer cómo funciona, para estudiarla, participar en su implementación o adecuarla a sus necesidades. Además, el incremento del **desarrollo de *software* libre, código abierto y *hardware* libre**, permite compartir inquietudes y conocimientos, sirviéndose de la red de redes, Internet, como principal vía de comunicación.

Motivación

Puesto que el uso de la informática viene de la mano de la aplicación de otras ciencias y hoy en día se aplica a todas ellas, con este trabajo se trata de mostrar cómo, eligiendo la tecnología apropiada y conociendo los límites de la misma, a través de un sencillo programa (al que llamaremos *sketch*, ya que es así como se conocen los programas en la plataforma Arduino), se puede conseguir,

sin necesidad de conocimientos electrónicos avanzados, una aplicación práctica como es la medición de distancias.

Tras una primera tarea de **investigación** para encontrar la tecnología más apropiada para la medición de distancias de manera sencilla, práctica y económica, se concluyó que lo más adecuado a nuestras necesidades era basarnos en la tecnología de Arduino. En concreto se utilizará un Arduino UNO y un sensor de ultrasonidos HC-SR04 compatible con él.

Arduino nació para enseñar Diseño de Interacción (*Interaction Desing*), es decir, el diseño de cualquier experiencia interactiva (entre humanos y objetos) mediante un proceso basado en la creación de prototipos de fidelidad en constante crecimiento, a través de la tecnología electrónica. Dentro del Diseño de interacción, el campo específico relacionado con Arduino es la Computación Física (*Physical Computing*) o Diseño de Interacción Física (*Physical Interaction Desing*), que se sirve de la electrónica para **crear prototipos** de materiales nuevos **para** que, diseñadores y artistas, puedan **crear objetos interactivos**, que se puedan comunicar **con** los humanos a través **sensores** y **actuadores** (o activadores) **controlados** por un ***software*** que se ejecute en un microcontrolador (un pequeño ordenador en un solo chip). Mediante Arduino, se pueden conocer los elementos básicos de la electrónica y los sensores rápidamente, creando prototipos sencillos, **sin demasiada inversión**.

El software libre, el código abierto y el hardware libre de Arduino

El *software* de Arduino es su conjunto lógico de instrucciones (programa), integrado en el entorno de desarrollo (IDE); el *hardware* es el conjunto de sus componentes físicos (la tarjeta en sí). La plataforma o sistema Arduino se basa en el *software* libre, en el código abierto y en el *hardware* libre (aunque el sensor HC-SR04, compatible con Arduino, no). Estos conceptos, que suelen estar relacionados, a veces dan lugar a confusión. A continuación se explican sus fundamentos básicos.

El ***software* libre** (*free software*) proporciona a sus usuarios (tanto de forma individual, como colectiva) la **libertad** permanente, irrevocable y sin justificación, **de ejecutarlo** en cualquier sistema y para cualquier propósito; de **estudiarlo** y ver cómo funciona; de **modificarlo** y **mejorarlo**, accediendo a su código fuente sin tener por qué notificarlo; **copiarlo**, **distribuirlo y publicarlo** (preferiblemente en forma de ejecutables, que suelen ser más sencillos de instalar) con su correspondiente código fuente, con o sin modificaciones, sin tener que pedir permiso o pagar por ello, y cobrando, o no, una tasa (porque aunque sea libre, no tiene por qué ser gratuito), para que el resto de usuarios (lo que se conoce como "la comunidad") continúen beneficiándose de sus revisiones y posteriores versiones y, de esta forma, **se pueda controlar** dicho *software* y lo que hace.

Como se ha explicado, para que un *software* sea ***software* libre**, **debe ser *software* de código abierto** (*open-source*). El código abierto, que surge a partir del concepto de *software* libre, busca ofrecer a todo el mundo el **acceso al código, permitiendo su modificación y redistribución**. El código fuente de Arduino se puede descargar o consultar de forma libre, desde su sitio web oficial[1].

Existen dos organismos no lucrativos a nivel mundial, la OSI (*Open Source Initiative*)[2] y la FSF (*Free Software Foundation*)[3], relacionados con el concepto de "*software* libre" (a partir del término anglosajón "*free software*") y el concepto de "*software* de código abierto" ("*open-source software*", en inglés). En defensa de los derechos de los usuarios del *software* libre, apoyan y fomentan el movimiento de código abierto, promoviendo licencias para su libre distribución (entre las que destacan las licencias GPL y LGPL, bajo las que se encuentra Arduino).

Para que, a efectos legales, se hable de *software* libre, éste ha de someterse a algún tipo de licencia de distribución que lo caracterice como tal. Algunas de las licencias de *software* libre (y *open-source*) más conocidas son: la licencia GPL, la CC y la BSD. La BSD (*Berkeley Software Distribution*)[4], permite trabajos derivados sin restricción alguna, de forma que el autor tan solo mantiene el reconocimiento por sus contribuciones. Pero Arduino, sin embargo, utiliza las otras dos (GPL y CC).

La licencia **GPL** (*General Public License*)[5], surgida del proyecto para el sistema operativo GNU (de la FSF), busca asegurar los derechos a la hora de utilizar, estudiar, compartir y modificar código, con la única obligación de que, el código derivado, utilice la misma licencia. Arduino consta de dos licencias de este tipo: la licencia GPL y la licencia LGPL.

- La **GPL o Licencia Pública General** (Versión 2, de junio de 1991. Copyright © 1989, 1991 *Free Software Foundation*, *Inc.*)[6], establece once términos y condiciones en cuanto a copia, distribución y modificación se refiere. Bajo esta licencia se encuentra el código de procesamiento que, de forma general, conforma el entorno visual de programación de Arduino.

- La licencia **LGNU o Licencia Pública Menos General** (Versión 2.1, de febrero de 1999. Copyright © 1991, 1999 *Free Software Foundation*, *Inc.*)[7], sucesora de la anterior, establece quince términos y condiciones para la copia, distribución y modificación. Bajo esta

[1] En la web de Arduino, está su código fuente: https://code.google.com/p/arduino/source/browse/#svn/trunk
[2] Sitio web oficial de la *Open Source Initiative* (OSI): http://opensource.org/.
[3] Sitio web oficial de la *Free Software Foundation* (FSF): https://www.fsf.org/.
[4] Para más información sobre las licencias BSD, se puede consultar: http://en.wikipedia.org/wiki/BSD_licenses.
[5] Las licencias de GNU *Operating System* pueden consultarse en: http://www.gnu.org/licenses/licenses.en.html
[6] La licencia GPL de Arduino se puede consultar en: http://www.gnu.org/licenses/gpl-2.0.html
[7] La licencia LGPL de Arduino se puede consultar en: http://www.gnu.org/licenses/lgpl-2.1.html

licencia se encuentran el resto de bibliotecas y paquetes, especialmente diseñados para el control del microcontrolador de Arduino a nivel más interno. Ambas se encuentran *on-line*[8], en el sitio web de Arduino y, en general, en el sitio web de GNU.

La **CC** (*Creative Commons*)[9] es una organización no lucrativa que establece un conjunto de licencias a través de las cuales, el autor de una obra puede seleccionar los derechos que quiera otorgar a sus usuarios, permitiendo o no su uso comercial, así como modificaciones de licencia en trabajos derivados. **Arduino** se encuentra bajo la **licencia Creative Commons (CC) BY-SA**. "BY" quiere decir "*Attribution*" (para el reconocimiento de la autoría); y "SA", "*Share-Alike*" (es decir, compartir de igual forma). Esto significa que permite realizar trabajos derivados, tanto personales como comerciales, siempre que éstos den crédito a Arduino y publiquen sus diseños bajo la misma licencia.

Hardware y software libre, imagen 1

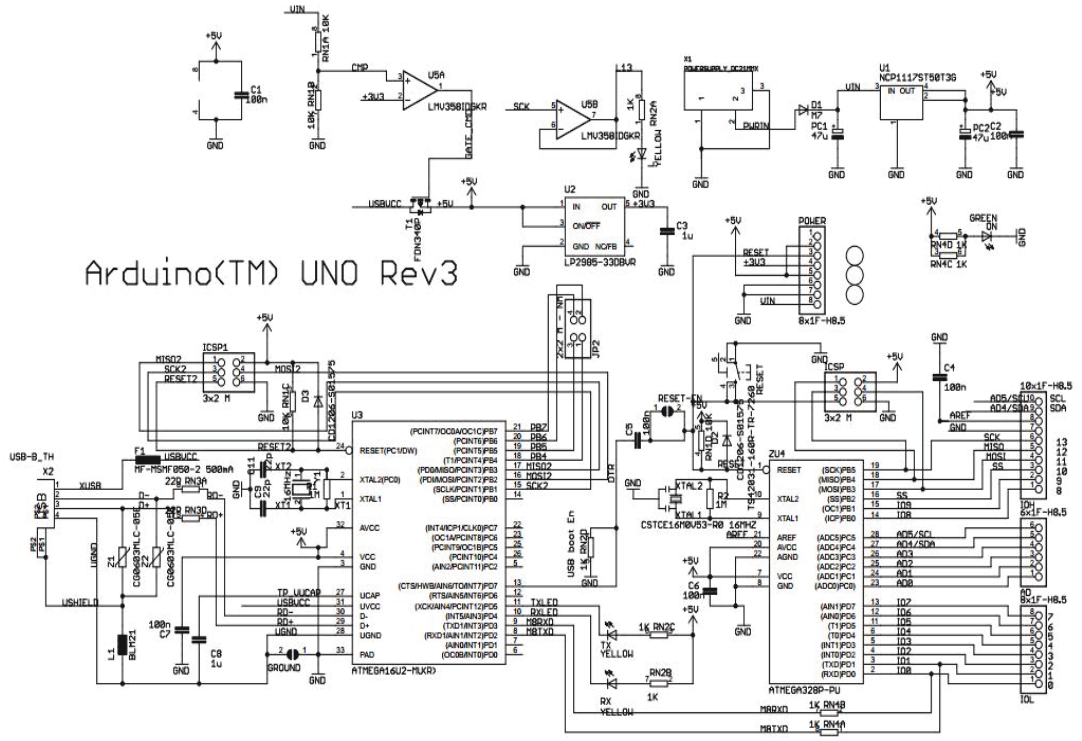
El ***hardware* libre** permite poder estudiar el *hardware*, **reutilizarlo**, **modificarlo**, **mejorarlo** y **compartirlo**. Su objetivo es facilitar y acercar a los usuarios la electrónica, la robótica y la tecnología actual en general, involucrándolos para entenderla, sacar mayor rendimiento de ella, y participar en la creación de futuras tecnologías. El *hardware* libre permite conocer qué hay dentro de las cosas de una forma éticamente correcta. Por ello, junto con el *software* libre es generador de conocimiento y, desde ese punto de vista, es importante para la electrónica, la informática y la vida en general. Además, de la misma forma que el *software* libre se basa en el código abierto, el *hardware* libre (aquél que viene con sus especificaciones técnicas, de modo que el usuario pueda crear él mismo una réplica de dicho *hardware*) **se basa**, inevitablemente, **en el *hardware* de diseño abierto** (aquél que viene con las especificaciones completas de manera que el usuario pueda interactuar con él sin ningún tipo de sorpresas desagradables y sin necesidad de saber lo que pasa dentro). Para hablar de *hardware* libre, **es necesario tener acceso a los ficheros esquemáticos del diseño** *hardware*, **con la información necesaria** (es decir, ha de proporcionar qué componentes individuales lo integran y qué interconexiones hay entre cada uno) **para que cualquier usuario** (con los materiales, herramientas y conocimiento adecuados), **pueda reconstruirlo** por su cuenta.

El *hardware* libre (a diferencia del software libre), **no tiene prácticamente licencias** bajo las que establecerse como tal. Para ello, suele ser denominado *free hardware design* u *open source harware*[10]; o se le puede aplicar algún tipo de licencia similar a las que existen para el *software*.

[8] Licencias GPL y LGPL de Arduino: https://code.google.com/p/arduino/source/browse/trunk/license.txt
[9] El sitio web de la CC, donde se pueden encontrar sus licencias, es: http://creativecommons.org/.
[10] Los conceptos de *"free hardware design"* y *"open-source hardware",* que vienen a significar lo mimo, se pueden consultar en: http://en.wikipedia.org/wiki/Open_source_hardware.

El proyecto **OSHW** (*Open Source HardWare*)[11] es una declaración de intenciones (no una licencia como tal) en forma de normas y características, que trata de establecer una colección de **principios** que ayuden a identificar a un producto físico como *hardware* libre y respeten la libertad de los creadores para controlar su tecnología, de forma que se puedan compartir conocimientos y se fomente el intercambio abierto de diseños, mostrando una **alternativa a las patentes** de *hardware*, para conseguir un *hardware* que se desarrolle bajo licencias de *hardware* libre.

La tarjeta o placa Arduino es ***hardware* libre**. La plataforma Arduino proporciona tanto la **documentación necesaria**, como los **ficheros** y esquemas de **diseño**, disponibles bajo la licencia Creative Commons (CC) BY-SA, 2.5 Generic[12]. Así, aunque lo habitual es adquirir la tarjeta Arduino de un distribuidor, ya ensamblada y lista para usar, un usuario tiene la posibilidad de fabricarse la suya propia, lo cual (al ser algo físico) cuesta dinero, por lo que, aunque sea libre, no es gratuita. El esquema[13] de Arduino UNO que se muestra a continuación y cuya última revisión es la Rev3, está disponible, junto con los diseños de referencia, en sitio web de Arduino.



Hardware y software libre, imagen 2

[11] Para mayor información sobre el proyecto OSHW, se puede consultar: http://freedomdefined.org/OSHW y http://www.oshwa.org/.

[12] La licencia CC BY-SA (2.5 Generic), bajo la que se encuentra Arduino, se puede consultar la CC, en los enlaces: http://creativecommons.org/licenses/by-sa/2.5/ ó http://creativecommons.org/licenses/by-sa/2.5/legalcode

[13] El hardware de Arduino, es libre. Sus diseños y esquemas de referencia se pueden consultar o descargar desde el sitio web de Arduino, en: http://arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf

Uno de los movimientos beneficiados por el código abierto, es el **DIY** (*Do It Yourself*), basado en la idea de crear cosas propias, a nivel tanto doméstico como industrial. Existen varios enfoques de este movimiento, dependiendo del ámbito de aplicación; a nivel educativo y como *hobby*, Arduino es una plataforma de referencia y la base para el desarrollo de nuevos sistemas. Además, en torno a un proyecto de código abierto, se puede formar una **comunidad** de personas que desinteresadamente mantienen, mejoran, corrigen, documentan o traducen el proyecto. Estas comunidades pueden estar más o menos organizadas y suelen utilizar diversas herramientas en Internet para comunicarse y compartir información. Arduino, por supuesto, también tiene la suya.

La comunidad de Arduino

Desde el punto de vista del desarrollo en común, Arduino no es simplemente una tarjeta programable, si no que forma parte de un proyecto o comunidad, que incluye, además del *hardware*, el *software* para la programación y comunicación con la placa; así como tutoriales y documentación que explican cómo aprender a utilizarlo, su propia *wiki*[14] (denominada *playground*) y diversos foros[15] para comunicarse y **compartir ideas o conocimientos** con más usuarios de Arduino.

Como ya se ha comentado anteriormente, en la web oficial de Arduino se puede acceder a la documentación del *software* y *hardware* y a repositorios (o almacenes) públicos, que alojan las distintas partes de Arduino, y desde los que se puede descargar su código fuente. La comunidad puede ser de gran utilidad, como apoyo adicional, en temas relacionados con Arduino. Hoy en día, además, se puede permanecer en continuo contacto con la comunidad de Arduino a través de Twitter[16], Google+[17], Facebook[18] y Youtube[19].

[14] La wiki de Arduino, conocida como *playground* es: http://playground.arduino.cc/.
[15] El foro oficial de Arduino es: http://forum.arduino.cc/.
[16] Puedes seguir a Arduino en Twitter, en el enlace: http://www.twitter.com/arduino.
[17] Puedes seguir a Arduino en Google+, en el enlace: https://plus.google.com/+Arduino/posts.
[18] Cuenta de Facebook de Arduino: https://www.facebook.com/official.arduino.
[19] Canal de Youtube de Arduino: http://www.youtube.com/user/arduinoteam.

Objetivos del Proyecto

Medición de distancias con Arduino.

La finalidad de este proyecto es conocer y trabajar con la tecnología de Arduino y del sensor de ultrasonidos HC-SR04 para **desarrollar una aplicación** práctica, como es la **medición de distancias**. Además, las medidas de la distancia que vayan siendo tomadas por la placa Arduino y el sensor, se han de visualizar, de forma sencilla, a través de un ordenador.

El proceso de desarrollo de la aplicación y la aplicación misma, se explicarán más adelante en este documento (concretamente en los apartados "Diseño del programa" e "Implementación del programa").

Análisis de prestaciones.

Con este proyecto, se pretende además, **mostrar gráficamente la precisión de la medición** de distancias para su futura aplicación en desarrollos de alto rendimiento. Por ello, se han llevado a cabo diversas pruebas que permiten visualizar dicha precisión y el grado de error que puede tener este sistema de medición. Esto, nos dará una idea más concreta, de en qué ámbitos en concreto se puede utilizar este sistema de medición, y cuáles son sus limitaciones.

El análisis de las prestaciones de "Arduino + Ultrasonidos", así como las pruebas y resultados obtenidos, se detallan posteriormente (en el apartado "Evaluación y Resultados").

Medios técnicos necesarios

Con un **presupuesto inferior a 50€** y un ordenador (con un sistema operativo Windows, Mac OS X o Linux) en el que ha de instalar el IDE Arduino, **es posible realizar la aplicación de medición** de distancias por ultrasonidos objetivo de este trabajo. En la realización de las pruebas, para tener como referencia una medida de distancia exacta a un punto, en vez de una cinta métrica convencional, se ha utilizado un medidor de distancias láser proporcionado por el GIAA de la Universidad.

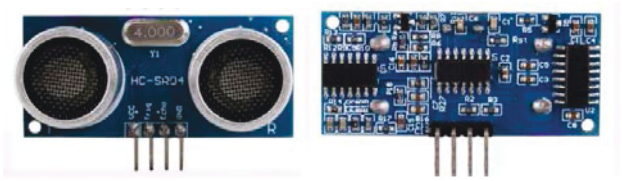
Arduino UNO

La última versión de esta tarjeta es la R3. Para un uso no experto se recomienda utilizar Arduino UNO R3 con microcontrolador en formato DIP (o PU), frente al SMD, por si se dañase el microcontrolador al trabajar con la placa (ya que simplemente bastaría con sustituirlo para seguir utilizando la placa). Arduino UNO R3 se puede adquirir a través del sitio web de Arduino, en su tienda oficial[20], o a través de los distribuidores oficiales[21] de Arduino. Su precio ronda los 25€.

Medios técnicos necesarios, imagen 1

Sensor de ultrasonidos HC-SR04

El sensor HC-SR04 es compatible con Arduino. Se puede adquirir también, mediante alguno de sus distribuidores oficiales. Cuesta unos 5€.

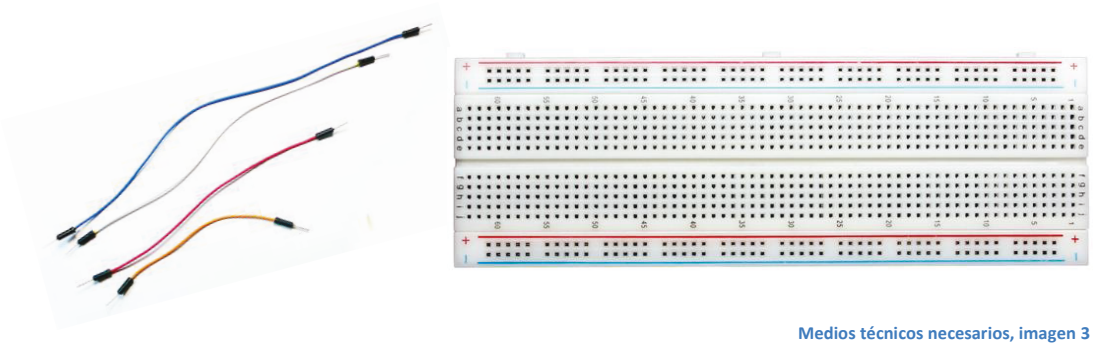


Medios técnicos necesarios, imagen 2

[20] Tienda oficial de Arduino: http://store.arduino.cc/.
[21] Distribuidores oficiales de Arduino: http://arduino.cc/en/Main/Buy.

Placa de prototipos y cables

Tanto la placa de prototipado como los cables eléctricos para las conexiones, se pueden encontrar, además de en la tienda oficial de Arduino o en alguno de sus distribuidores oficiales, en cualquier tienda de electrónica por unos 12€.



Medios técnicos necesarios, imagen 3

Ordenador

Es necesario un ordenador en el que se ha de descargar el IDE de Arduino, para comenzar a escribir los programas. Puede utilizarse cualquier PC con un sistema operativo Windows, Mac OS X o Linux. En este proyecto, se ha utilizado un PC portátil Samsung R510 con Windows 7 Professional (Copyright © 2009 Microsoft Corporation) de 32 bits, memoria RAM de 4[GB] y procesador Intel(R) Core(TM)2 Duo CPU P8600 @ 2'40[GHz] 2'40[GHZ].

Cable USB

Finalmente, se necesita un cable USB que, además de para intercambiar información entre el ordenador y la placa, también servirá para alimentarla eléctricamente. Sirve cualquier cable con conector USB (*Universal Serial Bus*) estándar, de tipo A/B, para la conexión de periféricos al PC. Cuesta unos 3€.

Medios técnicos necesarios, imagen 4

Planificación

En la siguiente tabla se muestra la planificación establecida para la realización de este proyecto, reflejando el número de horas al día estimadas de trabajo necesario, y durante cada cuántas semanas (entendidas como semanas de lunes a viernes, en horario lectivo).

	Semana 1	Semana 2	Semana 3	Semana 4	Semana5	Semana 6	Semana 7	Semana 8	Semana 9	Semana 10	Semana 11	Semana 12	Semana 13	Semana 14	Semana 15	Semana 16	Semana 17	Semana 18	Semana 19	Semana 20	Semana 21	Semana 22	Semana 23	Semana 24	Semana 25
Planteamiento de la idea (medición de distancias) y **elección de la tecnología adecuada** (Arduino y un sensor) para llevarla a cabo.	4 horas / día																								
Investigación sobre Arduino en general y en particular sobre la placa Arduino UNO y su compatibilidad e interacción con el sensor ultrasónico HC-SR04.		4 horas / día	4 horas / día	1 horas / día	1 horas / día	4 horas / día	4 horas / día	1 horas / día	1 horas / día	4 horas / día	4 horas / día	1 horas / día	½ horas / día	½ horas / día	2 horas / día	2 horas / día									
Planteamiento del presupuesto definitivo y posterior adquisición de los **medios técnicos** necesarios llevar a cabo el proyecto.													½ hora /día	½ hora /día											
Desarrollo de la aplicación de medición de distancias a través de ultrasonidos (es decir, implementación del programa, *sketch*, mediante el IDE Arduino).															2 horas / día	2 horas / día									
Realización de la **memoria** del TFG. Estructura y diseño del documento. Redacción. Adicción de imágenes, tablas y gráficos. Corrección de posibles erratas.				3 horas / día	3 horas / día			3 horas / día	3 horas / día			3 horas / día	3 horas / día	3 horas / día			4 horas / día	4 horas / día	4 horas / día	4 horas / día	4 horas / día	4 horas / día	4 horas / día	½ horas / día	½ horas / día
Experimentación en la medición de distancias por ultrasonidos desarrollada. Realización de las **pruebas** del sistema.																								½ horas / día	½ horas / día
TOTAL horas / día	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	2	2
TOTAL horas / semana	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	10	10
TOTAL horas TFG	**480**																								

Planificación 1. 1

Presupuesto

A continuación se plantea el presupuesto necesario para la realización del proyecto, en base a los recursos necesarios, tanto humanos como materiales, para llevarlo a cabo, y de acuerdo a la planificación planteada.

De forma general, el presupuesto estimado en relación a los recursos materiales necesarios, se puede resumir de la siguiente forma:

		Tiempo que se ha utilizado [meses]	Período de depreciación [meses]	Coste del equipo por unidad [€]	Coste imputable [€]
HW	Arduino UNO	5	60	25	2'083333333
	Sensor Ultrasónico HC-SR04	5	60	5	0'416666666
	Placa de prototipos y cables	5	60	12	1
	Cable USB	5	60	3	0'25
	Ordenador (PC) con Windows 7 Professional y Microsoft Office 2007 instalado	6	60	600	60
	(Sensor láser Skil)	0'5	60	90	0'75
SW	IDE Arduino v. 1.0.5	---	---	Gratuito	---
	Fritzing v. 0.9.0	---	---	Gratuito	---
	COSTE TOTAL				89'25€

Presupuesto 1. 1

Por otra parte, el presupuesto relacionado con los recursos humanos se ha subdividido en las principales fases que componen el trabajo:

- **Estudio de la viabilidad del sistema** a desarrollar: estimación del esfuerzo y análisis de los conocimientos necesarios para realizar el proyecto. Tiene que ver con el planteamiento de la idea la y elección de la tecnología adecuada (Arduino y un sensor) para llevarla a cabo.

- **Diseño del sistema**: líneas generales del proyecto, afianzamiento de conocimientos previos, e investigación y estudio exhaustivo para ampliarlos, según los requisitos del sistema. Se ha de llevar a cabo mediante la investigación sobre la plataforma Arduino en general, y en concreto sobre la tarjeta programable Arduino UNO, y el sensor ultrasónico HC-SR04.

- **Implementación** del sistema: definición del sistema, medios técnicos necesarios concretos para la aplicación de medición, y desarrollo del programa.

- **Experimentación y evaluación** del sistema: elección y realización de las pruebas precisas, y resultados obtenidos de forma gráfica.

- **Documentación** del sistema: realización de la **memoria.**

	Categoría	Horas invertidas	Precio / Hora [€]	Coste total [€]
Viabilidad del sistema	Diseñador	20	13'39	2.544'1
Diseño del sistema		170		
Implementación	Programador	20	15'17	303'4
Evaluación y pruebas	Analista	20	18,75	375
Documentación	Redactor	250	8'92	2.230
TOTAL		**480**	**56'23**	**5.452'1€**

Presupuesto 1. 2

Marco Regulador

A pesar de que, en lo que atañe a este proyecto (en cuanto aspectos legales se refiere) éste no se ve afectado directamente por las leyes vigentes a nivel estatal en cuanto a telecomunicaciones, protección de datos y de carácter medioambiental, si bien, de forma indirecta, cabe mencionar el marco regulador actual aplicable a cualquier proyecto de ingeniería informática.

Ley General de las Telecomunicaciones

La Ley 9/2014, de 9 de mayo, General de Telecomunicaciones[22], publicada en el BOE núm. 114, de 10/05/2014 y entrada en vigor desde el 10/05/2014, persigue, como uno de sus principales objetivos, recuperar la unidad de mercado en el sector de las telecomunicaciones, fundando procedimientos de coordinación y resolución de conflictos entre la legislación sectorial estatal y la legislación de las Administraciones competentes dictada en el ejercicio de sus competencias que pueda afectar al despliegue de redes y a la prestación de servicios.

Ley Orgánica de Protección de Datos

La Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal[23], publicada en el BOE núm. 298, de 14/12/1999 y entrada en vigor desde 14/01/2000 tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor e intimidad personal y familiar.

Ley de Responsabilidad Medioambiental

La Ley 26/2007, de 23 de octubre, de Responsabilidad Medioambiental[24], publicada en el BOE núm. 255, de 24/10/2007 y entrada en vigor desde 25/10/2007, regula la responsabilidad de los operadores de prevenir, evitar y reparar los daños medioambientales, de conformidad con el artículo 45 de la Constitución y con los principios de prevención y de que «quien contamina paga».

[22] Ley de las Telecomunicaciones: https://www.boe.es/buscar/act.php?id=BOE-A-2014-4950.
[23] Ley Orgánica de Protección de Datos: https://www.boe.es/buscar/act.php?id=BOE-A-1999-23750.
[24] La ley de Responsabilidad Medioambiental: https://www.boe.es/buscar/act.php?id=BOE-A-2007-18475.

Estructura del documento

Hasta aquí, a modo de introducción del trabajo, se han tratado de explicar los fundamentos básicos y los objetivos de este proyecto. Además de los medios materiales necesarios para llevarlo a cabo, la planificación del trabajo, el presupuesto asociado y el marco regulador actual.

Las notas al pie que se encuentran en algunos de los términos utilizados en la redacción de este documento, pueden servir de ayuda para aclararlos, puesto que en muchas de ellas se hace referencia al sitio web del que han sido extraídos y, en otras, se explica dónde poder encontrar más información al respecto. Cualquier enlace a páginas web, se muestra subrayado. Los términos anglosajones aparecen en cursiva. Y las referencias a las funciones del lenguaje de programación se muestran con otro tipo de letra y color diferente, para poder diferenciarlas del texto normal.

A continuación se presenta una breve descripción de los sucesivos apartados que conforman este proyecto.

En el apartado denominado "**Arduino**", se comenzará dando una visión general sobre la plataforma Arduino explicando sus principales **características** *hardware* y *software*. Conforme avance el apartado, se profundizará más en la explicación del modelo de placa Arduino que se ha utilizado, el **Arduino UNO R3**. La mayor parte de la información sobre la plataforma Arduino en general, y sobre la placa Arduino UNO R3 en particular, ha sido extraída del sitio web oficial de Arduino.

Después, en el apartado denominado "**Sensor ultrasónico**" se explicarán tanto el concepto de **sensor**, como el concepto de **ultrasonidos**. Y se entrará en detalle en la explicación de las características y el funcionamiento interno del sensor de ultrasonidos utilizado en este proyecto, el sensor **HC-SR04**, que es compatible con Arduino.

Más adelante, en el apartado "**Diseño del programa**" se definirá el **sistema** a implementar, teniendo en cuenta el **alcance** y las **restricciones** del mismo. Se mostrará además, el esquema del **circuito** que forman la placa Arduino UNO R3 junto al sensor HC-SR04. Y finalmente se describirá el entorno operacional en que puede aplicarse.

A continuación, en el apartado "**Implementación del Programa**" se explicará todo el proceso que se ha seguido para llevar a cabo la aplicación de medición de distancias. Desde la **instalación del IDE**, hasta la **elaboración del programa** y los detalles de **comunicación entre** la placa **Arduino** UNO R3, el **ordenador con el IDE** de Arduino instalado, **y el sensor** ultrasónico HC-SR04.

Seguidamente, en los apartados "**Evaluación y resultados**" y "**Conclusiones**" se mostrarán las pruebas realizadas y los resultados obtenidos, que darán una idea aproximada en cuanto al grado de error que puede haber en la medición de unas u otras distancias, permitiendo reflexionar sobre la funcionalidad de la aplicación desarrollada, sus limitaciones y su potencial aplicación en desarrollos de alto rendimiento. Además se dará una idea de posibles futuros trabajos, a partir de este.

Para concluir el documento, se ha añadido un **anexo sobre el lenguaje de programación de Arduino**, un **anexo sobre los distintos tipos de sensores** y sus clasficaciones más comunes, y **anexo sobre los sistemas electrónicos y su proceso de producción**, un

Finalmente, en el apartado "**Bibliografía**" se pueden encontrar las referencias bibliográficas a los principales manuales, libros y sitios web que se han consultado a los largo de este trabajo.

ARDUINO

Arduino es una **plataforma libre** de circuito integrado y sistema informático, para la creación de prototipos electrónicos. Consta de una **parte hardware** (la propia placa) **y** una **parte software** (en forma de entorno de programación integrado o IDE, *Integration Development Environment*), ambas **flexibles y sencillas de usar**. Surgió de la idea de un ingeniero italiano (Guianluca Martino, en 2005) con fines docentes y, quizá debido a su **precio asequible** y a su carácter *open-source* (que permite su diseño y distribución libre), su uso se ha extendido en los últimos años a múltiples proyectos, muchos de ellos relacionados con el diseño industrial y el arte interactivo. La información oficial de Arduino se puede encontrar *on-line*, en su sitio web oficial.

Existen en el mercado distintos modelos de placas Arduino oficiales. Todas ellas, incluyen como elemento principal **microcontroladores** de la familia tecnológica **AVR** (de la marca Atmel), por lo que su funcionamiento interno es similar. Los pines del microcontrolador, se encuentran unidos internamente a los pines de la placa. Así, a través de la placa y del IDE Arduino, **mediante sensores** (como el de ultrasonidos utilizado en este proyecto) se puede **extraer información** del mundo físico, es decir, del entorno que nos rodea (para ello, las entradas o "pines de entrada" de la placa, reciben impulsos eléctricos que se transforman en corriente eléctrica); además, **mediante actuadores o activadores** (como *leds*, altavoces, *displays*, motores o pantallas) se puede **proporcionar información** (por eso, las salidas o "pines de salida" de Arduino ofrecen valores eléctricos). Algunos modelos de placas Arduino tienen sensores y actuadores integrados pero, **en general** (y así ocurre también con la placa Arduino UNO), **se trabaja con sensores y actuadores externos, y con una placa de prototipos** que simula el futuro circuito impreso, y en la cual se pueden montar y desmontar fácilmente las conexiones mediante cables. Además, a los distintos modelos de placas Arduino, se les pueden añadir *shields* (placas complementarias que mantienen la misma disposición que los pines de las placas Arduino, para apilarlos cómodamente sobre ellas, simplificando conexiones más complicadas). Existen *shields* de Ethernet, GPS o pantallas TFT, entre otros.

Las placas, pueden construirse a mano o comprarse ya preensambladas y el IDE oficial para trabajar con Arduino, puede descargarse de forma gratuita. Además, los desarrollos con Arduino pueden requerir de un sistema operativo para ejecutarse, o no (*stand-alone*), o incluso interactuar con un programa que se está ejecutando en un ordenador (Flash, por ejemplo).

Hardware

El *hardware* es el conjunto de componentes que integran la parte material de Arduino, es decir, **la propia placa** de Arduino. Se trata de una tarjeta **programable**, en forma de circuito impreso (PCB, *Printed Circuit Board*), cuya superficie está fabricada de un material no conductor (resinas de fibra de vidrio reforzada, cerámica o plástico) y trazada con pistas de un material conductor (cobre, normalmente) que conectan eléctricamente los distintos componentes soldados a la placa, formando un determinado diseño de circuitería interna, cuyo elemento principal es el **microcontrolador** (de Atmel) de 8 bits de resolución (a excepción del microcontrolador de la placa Arduino DUE, que es de 32 bits). Consta de **tres tipos de memoria**, de capacidad variable según el tipo de microcontrolador (en función del modelo de placa): memoria **Flash**, con una capacidad de entre **16** y **256[KB]**, memoria **SRAM** (*Static Random Access Memory*) de **1, 2 u 8[KB]**, y memoria **EEPROM** (*Electrically Erasable Programable Read-Only Memory*) de **512 bytes**, 1024 bytes (ó **1[KB]**) **ó 4[KB]**. El microcontrolador tiene una serie de conectores o **pines** que se utilizan como entrada o salida, **registros** de memoria, un **oscilador** interno (que facilita al microcontrolador una serie de pulsos para mantener estable la velocidad de trabajo), e **interfaces** para la comunicación serie; entre sus principales componentes.

Por otra parte, la placa Arduino consta también de una serie de entradas y salidas (en forma de conectores o pines), tanto analógicas como digitales. Los pines de **entrada o salida digital**, pueden comportarse, como su propio nombre indica, como entradas o salidas digitales, según se especifique en el *sketch* programado en el IDE; las **entradas analógicas** aceptan valores analógicos (es decir, lecturas de voltaje desde un sensor) y los convierten en un valor comprendido entre 0 y 1023; y las **salidas analógicas** son salidas digitales reprogramadas para ofrecer valores de salida analógicos mediante el *sketch* creado en el IDE.

La placa, para funcionar, necesita **alimentación eléctrica**. La alimentación de la placa puede provenir, bien de una fuente de alimentación externa (la red eléctrica general o una batería) o bien desde el computador al que esté conectada la placa mediante un cable USB. Si no hay una fuente de alimentación externa, la alimentación vendrá por el USB, pero en cuanto se enchufe una fuente de alimentación, la placa la usará automáticamente.

El diseño original de Arduino incluye los componentes electrónicos necesarios para alimentar y comunicarse con el microcontrolador así que, otras alternativas de diseño deberían incorporarlos o

al menos, mostrar la forma de añadirlos fácilmente. Los diseños de referencia de Arduino, como ya se ha comentado anteriormente, se encuentran bajo la licencia de CC BY-SA (2.5 Generic)[25].

A continuación se muestra una tabla[26], a modo de resumen de los principales componentes *hardware* de cada uno de los distintos modelos de placas Arduino:

Modelo de placa Arduino	Procesador, o tipo de microcontrolador	Tensión de funcionamiento [V]	Voltaje de entrada [V]	Velocidad de la CPU [MHz]	Entradas analógicas / Salidas analógicas	Entradas o salidas digitales / Salidas PWM	Memoria EEPROM [KB]	Memoria SRAM [KB]	Memoria Flash [KB]	Tipo de conexión USB	Nº de puertos serie UART
UNO	ATmega328	5	7-12	16	6/0	14/6	1	2	32	Regular	1
Due	AT91SAM3X8E	3.3	7-12	84	12/2	54/12	-	96	512	2 Micro	4
Leonardo	ATmega32u4	5	7-12	16	12/0	20/7	1	2.5	32	Micro	1
Mega 2560	ATmega2560	5	7-12	16	16/0	54/15	4	8	256	Regular	4
Mega ADK	ATmega2560	5	7-12	16	16/0	54/15	4	8	256	Regular	4
Micro	ATmega32u4	5	7-12	16	12/0	20/7	1	2.5	32	Micro	1
Mini	ATmega328	5	7-9	16	8/0	14/6	1	2	32	-	-
Nano	ATmega168	5	7-9	16	8/0	14/6	1	2	32	-	-
Nano	ATmega328	5	7-9	16	8/0	14/6	1	2	32	Mini-B	1
Ethernet	ATmega328	5	7-12	16	6/0	14/4	1	2	32	Regular	-
Esplora	ATmega32u4	5	7-12	16	-	-	1	2.5	32	Micro	-
ArduinoBT	ATmega328	5	2.5-12	16	6/0	14/6	1	2	32	-	1
Fio	ATmega328P	3.3	3.7-7	8	8/0	14/6	1	2	32	Mini	1
Pro (168)	ATmega168	3.3	3.35-12	8	6/0	14/6	0.512	1	16	-	1
Pro (328)	ATmega328	5	5-12	16	6/0	14/6	1	2	32	-	1
Pro Mini	ATmega168	3.3	3.35-12	8	6/0	14/6	0.512	1	16	-	1
LilyPad	ATmega168V	5	5-12	16							
LilyPad	ATmega328V	2.7-5.5	2.7-5.5	8	6/0	14/6	0.512	1	16	-	1
LilyPad USB	ATmega32u4	3.3	3.8-5	8	4/0	9/4	1	2.5	32	Micro	-
LilyPad Simple	ATmega328	2.7-5.5	2.7-5.5	8	4/0	9/4	1	2	32	-	-
LilyPad SimpleSnap	ATmega328	2.7-5.5	2.7-5.5	8	4/0	9/4	1	2	32	-	-

Arduino, imagen 1

Como ya se ha comentado anteriormente, es posible ampliar las características de una placa, añadiendo un *shield* (otra tarjeta de circuito impreso que se puede colocar sobre la parte superior de una placa de Arduino, apilándola sobre sus pines, sin necesidad de cables). Normalmente, los *shields* comparten las líneas GND, 5V (o 3V3), RESET y AREF con la placa, y suelen utilizar alguno de sus pines para comunicarse con ella. Se ha de tener en cuenta tanto la tensión del funcionamiento e intensidad de corriente que necesita el *shield* ya que, en función de sus características, puede llegar a consumir bastante corriente (unos 300[mA]) de la que le llega a la placa Arduino (unos 500[mA]).

[25] Para consultar la licencia CC BY-SA (2.5), se puede ir a: http://creativecommons.org/licenses/by-sa/2.5/.
[26] La información de esta tabla ha sido extraída directamente de la web de Arduino. Se puede consultar en el siguiente enlace: http://arduino.cc/en/Products.Compare

Software

En cuanto a la parte *software*, para programar el *hardware* de Arduino (indicándole qué debe hacer y bajo qué condiciones, y que así, proporcione salidas o reaccione ante entradas), es necesario un entorno de desarrollo o **IDE** (*Integration Development Environment*) en forma de un conjunto de herramientas *software* para desarrollar y probar los programas, denominados *sketches* en Arduino. **El IDE oficial** de Arduino **se basa en** el lenguaje de programación de código abierto **Processing**[27], diseñado especialmente para personas sin experiencia en programación; **y en** la plataforma **Wiring**[28], para el desarrollo sencillo de prototipos de aplicaciones. Este IDE oficial, **puede descargarse** desde el sitio web de Arduino **de forma gratuita**[29]**. Existen otros IDEs alternativos** no oficiales que también podrían ser utilizados para trabajar con Arduino (como son: CodeBlocks, Gnoduino, Codebender, Visualmicro, EmbedXcode, Scratch for Arduino -S4A-, ModkitMicro, Minibloq, o Ardblock, entre otros). Aunque no será el caso de este proyecto, en el cual se ha utilizado el IDE oficial, por ser el más estandarizado para Arduino.

El **lenguaje de programación** de Arduino[30] se conoce como *Arduino Programming Language*. Está basado en el lenguaje C, por lo que soporta cualquier construcción estándar de C (y algunas de las funcionalidades de C++). Enlaza con la librería Libc AVR, permitiendo la utilización de cualquiera de sus funciones. A grandes rasgos, el lenguaje de programación de Arduino, se puede dividir en tres grandes grupos: **estructuras o bloques** (como `setup()`, `loop()`, operadores, etc.), **valores** (variables y constantes) y **funciones**. Existen además, gran número de librerías[31] para la interacción de Arduino con determinados tipos de *hardware*. Y numerosas contribuciones de código por parte la comunidad.

De esta forma, la placa se puede programar para que interactúe con el mundo real. Pero las **salidas** que ofrece la placa Arduino **y** las **entradas** ante las que puede reaccionar, son **de naturaleza eléctrica**. Por eso, **a la hora de enviar o captar señales** (ultrasonidos, en el caso de este proyecto) a través de sensores externos (el sensor HC-SR04, en este caso), habrá que **transformar el valor** leído por el sensor **en un valor eléctrico, procesarlo** mediante Arduino **y transformarlo** de nuevo, **en algo**

[27] Para obtener más información al respecto, el sitio web de Processing es: http://processing.org/.
Las principales diferencias entre Processing y el *Arduino Programming Language* pueden encontrarse en: http://arduino.cc/es/Reference/Comparison.
[28] Wiring también es una plataforma *open-source.* Fue diseñada para la programación de microcontroladores, y también está basado en Proccessing. El sitio web de Wiring es: http://wiring.org.co/.
[29] El IDE Arduino oficial está disponible en: http://arduino.cc/en/Main/Software.
[30] Para conocer más detalles sobre el lenguaje de programación en Arduino, se puede consultar el ANEXO I de este documento. O acceder al sitio web de referencia, en: http://arduino.cc/en/Reference/HomePage.
[31] Las librerías de Arduino, también se pueden encontrar en: http://arduino.cc/es/Reference/Libraries

utilizable para nosotros (que en el caso de este proyecto, será una distancia en centímetros, que aparecerá en el monitor serial del IDE instalado en el ordenador con el que se trabaja).

Por otro lado, en el IDE y de forma transparente al programador, el proceso de **compilación** que realiza interiormente Arduino consiste en la **transformación del programa en lenguaje C++**, creado a partir del código en lenguaje Arduino (programado en el IDE), **en el programa en código máquina** (binario) **AVR** (con la extensión ***.hex**) **ejecutable** por el microcontrolador. El compilador que incluye el IDE de Arduino[32] y que realiza dicha transformación es la herramienta "**gcc-avr**", una variante del compilador "gcc" para microcontroladores AVR, con una extensa librería de funciones que pueden ser utilizadas en Arduino. El proceso de compilación completo en Arduino se explica en las próximas líneas.[33]

En cualquier proceso de compilación (o de traducción de un programa a un lenguaje máquina ejecutable) se suceden dos etapas: una primera (de análisis), para dividir un programa fuente en los elementos que lo componen y crear un programa intermedio; y una segunda etapa (de síntesis), para construir el programa objeto que será transformado, mediante un *linker,* en el programa ejecutable.

En el núcleo de Arduino, el *toolchain* es la secuencia de programas[34] que se invocan cada vez que se compila un proyecto, generando el fichero de código máquina con extensión *.hex que será enviado a la placa Arduino (a su micro) a través de la conexión USB o serie (o un programador ISP, si no tuviese el *bootloader* instalado). Durante el proceso de compilación de Arduino, **se compilan** tanto **el *sketch*** (o los *sketches*) del proyecto con extensión *.ino, **como las librerías** incluidas, **y los ficheros que utiliza Arduino.** Los ficheros "*.ino" se fusionan (en un "*main sketch file*") antes de ser enviados al compilador avr-gcc, mientras que los ficheros "*.c" y "*.ccp", son compilados por separado.

En primer lugar, se ejecuta el **preprocesador** de Processing, que transforma el **código fuente** del lenguaje Arduino **en lenguaje C++**. Las **estructuras** `setup()` y `loop()` del programa en Arduino, se convierten en funciones de un programa C++ y se guardan en un **fichero** "***.ccp**" en el directorio

[32] La configuración completa del compilador, si se quiere consultar, se encuentra en los ficheros "makefile".
[33] Más información sobre compilación de Arduino: http://arduino.cc/en/pmwiki.php?n=Hacking/BuildProcess.
[34] Los detalles del *toolchain* (comandos que se ejecutan y directorio temporal del proyecto) se pueden hacer visibles activando la salida detallada de la consola del IDE cuanto a verificación y carga de programas. Se encuentran en "Archivo\Preferencias". Internamente en el IDE, estos procesos se lanzan desde los ficheros de la carpeta "app\src\processing\app\debug".

temporal[35] del proyecto, cuya ruta se puede hacer visible en la consola del IDE activando la salida detallada.

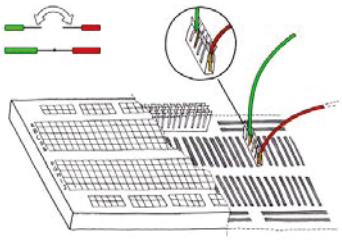
Después, se ejecuta el **compilador** "avr-gcc". Se incluyen en el *path* todas las rutas necesarias (el directorio del *sketch*, el directorio de destino[36], el directorio "include avr"[37] y los directorios de librerías). El compilador **analiza el texto del fichero "*.ccp"** (al cual se le añaden el resto de ficheros con extensión *.ccp, *.c ó *.s) **y del resto ficheros incluidos en el *path***. **Compila el *sketch*, las librerías y núcleo**. Cada fichero "*.ccp" o "*.c" se compila (con "avr-g++") como un objeto fichero distinto y se generan varios ficheros "*.o" en el directorio temporal del proyecto.

Más tarde, el programa ***linker***[38] une todos los ficheros objeto y los ubica en las posiciones que ocuparán en la memoria. Mediante "avr-ar", **une los ficheros "*.o" y los comprime en un fichero "*.a"**; y mediante "avr-g++", **genera un fichero de librería estática, "*.elf"** (un estándar de los ficheros ejecutables en UNIX) con todo código objeto generado antes. El fichero "*.elf" **se convierte** (a través de "avr-objcopy") **en un fichero "*.hex"** (o Intel Hex Format[39]) que contiene el **código máquina** que se cargará y ejecutará **en el microcontrolador**. Finalmente, **se envía el fichero** final (con extensión ***.hex**) **a la placa** Arduino.

El *bootloader*, que **es lo primero en ejecutarse** al iniciar la placa Arduino, espera un tiempo a que le llegue el fichero "*.hex". Si no le llega, pasa a la ejecución del programa principal; y si le llega, comienza a grabar el programa en la memoria Flash. Al cargar (tras verifcar) un *sketch* en el IDE, el programa "avrdude" se comunica mediante un pequeño protocolo con el *bootloader* a través del puerto USB. En la placa Arduino se realiza la traducción USB a serie y el programa "avrdude" envía el fichero "*hex." al *bootloader* que, como tiene permiso para escribir en la memoria Flash (de programa) del microcontrolador, la envía los datos que le ha pasado el "avrdude"; y cuando termina, reinicia la CPU.

[35] El código principal del programa que será construido en C++, se encuentra en la carpeta de instalación de Arduino, en "hardware\arduino\cores\arduino\main.ccp".
[36] El directorio de destino será "Arduino\hardware\arduino\cores\core"
[37] El directorio "include avr" se encuentra en "Arduino\hardware\tool\avr\avr\include\"
[38] El *linker* (o enlazador) de un proceso de compilación, es el programa encargado de unir al programa objeto, el código máquina de las funciones de las librerías utilizadas en el programa.
[39] El fichero "*.hex" generado a partir del proceso de compilación, contiene líneas de texto (cada una de las cuales contiene el número de bytes de línea, la dirección de comienzo de los datos en la Flash que les siguen, un *flag*, los datos con el código máquina que se debe programar y un código de comprobación de errores).

Placa de prototipos



Medios técnicos necesarios, imagen 5

La placa de prototipos, también conocida como *protoboard*, *breadboard* o *plugboard* es una placa de plástico con orificios, cada uno de los cuales tiene un contacto de resorte y una distancia de 2'54[mm] entre sí. Se utiliza como una placa de pruebas sin soldadura en la cual se pueden insertar componentes y extremos de cables, para simular un circuito impreso.

Los **orificios interiores** de la *protoboard*, normalmente están **divididos en dos bloques** (un bloque sería el formado por las líneas verticales A,B,C,D,E; y el otro por las líneas verticales F,G,H,I,J) y **están conectados entre sí en línea vertical**, de forma que, si se aplica una tensión a un agujero de un bloque, insertando uno de los contactos del componente en uno de los orificios de la *protoboard*, **todos los orificios de la misma línea vertical** (de ese bloque) **tendrán la misma tensión**. Los bloques se encuentran separados entre sí eléctricamente de manera que, cuando se conecta un chip, no se produzca un cortocircuito.

Las filas superior e inferior de los bordes horizontales de la *protoboard* (que suelen ser de color rojo y azul, y normalmente están marcados con los signos + y -, según su polaridad) **son los buses de alimentación**. Se conectan horizontalmente para llevar energía por la placa de prototipos de forma que, cuando se necesita **potencia** (alimentación eléctrica) **o tierra**, se pueda proporcionar rápidamente con un cable que conecte dos puntos del circuito.

Arduino UNO

Arduino UNO es una de las placas de Arduino más extendidas comercialmente y, por tanto, la elegida para este proyecto. De entre todas las variedades de placas Arduino[40] que existen en el mercado (UNO, Due, Leonardo, Mega, Micro, Mini, Nano, Ethernet, Esplora, Fio, Pro, Pro Mini, Lilipad, Yún, Tre y Robot) el modelo más estandarizado y utilizado es Arduino UNO. La que aquí se utiliza, en concreto, es la tarjeta programable Arduino UNO R3. "R3" quiere decir que es la tercera revisión de la placa, la última de la gama UNO.

Se han sucedido **múltiples revisiones de la placa** Arduino **en la gama Arduino USB**, **desde** que apareciese **la primera** placa denominada Arduino (**USB Arduino**) la cual ha ido evolucionando (desde V2.0 USB Arduino, Arduino Extreme, Arduino Extreme V2, Arduino NG, Arduino NG RevC, Arduino Decimila y Arduino Duemilanove) **hasta** llegar a **la última, Arduino UNO**, el último modelo de placas con USB y, según su sitio web oficial, el modelo de referencia, junto con el futuro Arduino 1.0 para la plataforma Arduino. Arduino UNO se diferencia del resto de placas de la gama USB anteriores en que no utiliza el controlador FTDI de USB a serie, sino que **cuenta con el chip Atmega16U2** (Atmega8U2, hasta la versión Arduino UNO R2) programado **como un traductor de USB a serie**. Desde su aparición en 2010, Arduino UNO **ha sufrido 3 revisiones**. La placa que se ha utilizado para este proyecto será **la última, Arduino UNO R3**. Por tratarse de la última revisión, la placa Arduino UNO R3 tiene algunas ventajas, además de nuevas etiquetas visuales para identificar las entradas y salidas fácilmente, respecto a versiones anteriores (pines de **salida SDA y SCL** cerca del pin AEREF, dos nuevos **pines** cerca del pin RESET, una instrucción **IOREF** que permite a los *shields* adaptarse al voltaje que suministra la placa, un circuito ***reset* consistente**, y el chip **Atmega16U2** en vez del Atmega8U2).

[40] En el sitio web de oficial de Arduino, se encuentra publicada la información de los distintos modelos de placas Arduino. Para el modelo Arduino UNO R3 en https://www.arduino.cc/en/Main/ArduinoBoardUno y para consultar cualquier otro modelo puede ir a https://www.arduino.cc/en/Main/Boards.

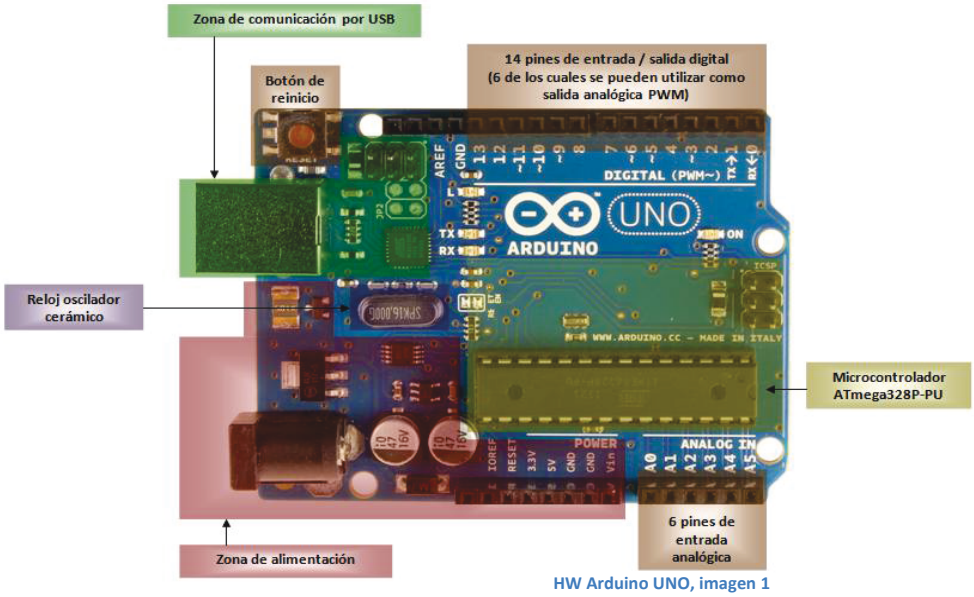
Hardware

Arduino UNO consta de un **microcontrolador Atmega328P**. La tarjeta programable contiene todo lo necesario para apoyar el microcontrolador, basta con conectarla al computador mediante un cable y conector USB para empezar a programar y a utilizarla. Se puede ampliar su funcionalidad con gran variedad de *shields* para funciones específicas.

La longitud y anchura de la placa Arduino UNO son unas 2'7 y 2'1 pulgadas respectivamente, excepto por la zona del conector USB y de alimentación, que sobresale unos milímetros. La distancia entre los pines digitales es de 0'1 pulgadas (excepto entre los pines 7 y 8 que es de 0'16['']). Tiene 4 orificios para poder atornillar la placa a alguna superficie. Su **tensión de funcionamiento** son **5[V]**, con un **voltaje de entrada** recomendado entre los **7 y** los **12[V]** (los límites teóricos son 6 y 20[V]) y una **intensidad de corriente DC máxima de los pines de E/S de 40[mA]** (50[mA] para el pin 3.3V).

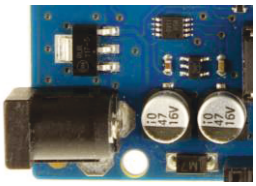
La placa cuenta con, además del microcontrolador **ATmega328P** con su correspondiente **memoria** (Flash de 32K, SRAM de 2K, y EEPROM de 1K) y cabezal **ISCP**, con una zona preparada para conexión de **alimentación**, otra zona para la conexión **USB**, y una serie de **conectores (pines) de entrada salida:** 14 pines digitales de E/S (6 de los cuales pueden utilizarse como salidas analógicas PWM) y 6 pines analógicas de entrada. También incluye un **resonador cerámico** con frecuencia de **16[MHz]**, un botón de ***reset*** (y varias formas de reiniciar la placa), interfaces para la comunicación con UART, I^2C (o TWI) y SPI 14 y otro microcontrolador (el ATmega16U2), con otro ISCP. Entre otros.

A continuación se muestra una imagen de la placa, con las principales parte diferenciadas. En las próximas líneas, se describen con más detalle.



HW Arduino UNO, imagen 1

La alimentación eléctrica


HW Arduino UNO, imagen 2

El **voltaje (alimentación eléctrica)** necesario para el funcionamiento de la placa Arduino son **5[V]**. Como se explica más detalladamente en los próximos párrafos, esta tensión se puede obtener mediante la conexión de la placa a una fuente de **alimentación externa o** utilizando un cable **USB** que conecte la placa con el computador. Respetando los **límites** de **entre 6 y 20[V]**, Arduino UNO es capaz de detectar cuál es el origen de la fuente de alimentación y seleccionar automáticamente la que esté disponible en cada momento.

Una de las formas de obtener alimentación eléctrica para la placa, es mediante la **conexión de la placa a una fuente de alimentación externa**. La placa está, teóricamente, preparada para recibir un voltaje de entrada de 6 a 20[V] aunque, en la práctica, para mayor estabilidad y seguridad eléctrica de los circuitos, se recomienda que esta diferencia de potencial oscile **entre los 7 y los 12[V]** (que internamente se reducirán a 5[V], mediante el circuito regulador de tensión de la placa). La fuente de alimentación externa puede ser un adaptador AC/DC o una batería.

Un **adaptador AC/DC** (que se puede enchufar a través del zócalo de la placa Arduino para un conector jack) es un transformador del voltaje alterno (AC) ofrecido por la red eléctrica general, en voltaje continuo (DC) constante. El adaptador AC/DC que se conecte a Arduino UNO ha de tener las siguientes características:

- **Voltaje de salida** para el adaptador (y por lo tanto, voltaje de entrada para la placa) **de 7 a 12[V] DC**. En caso de que sea inferior a 7[V] puede que el pin de salida de 5V ofrezca menos de este valor y hacer que la placa sea inestable. Por otra parte, a pesar de que el circuito de la placa es capaz de manejar hasta 20[V] (es decir, se podrían utilizar adaptadores AC/DC con salida de 20[V] DC) como gran cantidad del voltaje se pierde en forma de calor, si se suministran más de 12[V], el regulador puede sobrecalentarse y deteriorar la placa.

- **Intensidad de corriente ofrecida, 250[mA] o superior**. Al conectar a la placa una gran cantidad de componentes, o pocos pero que consuman mucha energía (como un motor, una tarjeta SD o una matriz de LEDs), el adaptador debería suministrar al menos, 500[mA] o incluso 1[A] para que haya suficiente corriente como para que cada componente funcione de forma estable.

- Polaridad positiva en el centro del conector jack, es decir que, la parte externa del cilindro metálico que lo forma, sea el borne negativo y, el hueco interior del cilindro, sea el borne positivo.

Por otro lado, como alternativa al adaptador AC/DC y como fuente de alimentación externa para la placa, se puede optar por una utilizar una batería (**pila**). Los cables salientes de los bornes (positivo y negativo) de la pila habrán de conectarse a los pines-hembra "**Vin**" (para el **polo positivo**) y "**GND**" (para el **negativo**), que se encuentran la zona de la placa etiquetada como "POWER". Una buena elección es una pila de 9[V], ya que ofrece un voltaje dentro el rango aconsejable de 7 a 12[V].

La otra alternativa de alimentación eléctrica para la placa Arduino UNO, es a través de un **cable USB** que la conecte **con el computador**. La placa consta de un conector USB hembra (tipo B) para introducir un cable con un conector USB macho (tipo A). La **alimentación** que recibe la placa a través del USB está **regulada permanentemente a 5[V]** y la **intensidad de corriente máxima** que ofrece son **500[mA]** (por lo que, sabiendo que la Ley de Ohm es V = I*R y aplicando la fórmula P = V*I, se puede deducir que, la potencia consumida por la placa, en ese caso, sería aproximadamente de P = V*I = 5*0'5 = 2'5[W]). La placa consta además, de un **polifusible reseteable** que protege los puertos USB del computador de cortocircuitos y sobrecorriente cuando se conecta a la placa el USB (aunque la mayoría de ordenadores también tienen su propia protección interna), ya que **rompe la conexión** (hasta que las condiciones eléctricas vuelvan a ser normales) **si llega a la placa más** intensidad **de** la deseable (**500[mA]**); el inconveniente de esta protección contra picos de corriente es que la intensidad de corriente recibida a través del cable USB puede no ser suficiente en proyectos con componentes que consuman mucha potencia (como motores y matrices de LEDs).


HW Arduino UNO, imagen 3

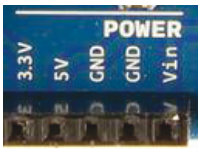
Cabe destacar que, la alimentación eléctrica en Arduino UNO, se basa en **un regulador del voltaje** (tipo "NCP1117ST50T3G" y que en los diagramas del circuito es nombrado "U1") **que** mantiene una tensión estable **de 5[V]** a su salida y puede suministrar hasta **1[A] de intensidad.** Existe además en la placa, **otro regulador** de tensión (tipo "LP2985-33DBVR" y nombrado como "U2") para conseguir una señal **de 3'3[V]** y hasta **150[mA]**, utilizada por muchos chips (como dispositivos externos y *shields*), en vez de la de 5[V].

Por otra parte, al lado del conector para la alimentación, se encuentran **dos condensadores** electrolíticos de **47[µF]**, para estabilizar la tensión y poder suministrar corriente de forma rápida ante demandas instantáneas, y **un condensador** electrolítico de **10[µF]**, para filtrar los posibles ruidos de

alta frecuencia. Junto a los condensadores electrolíticos de 47[µF] se sitúa un **diodo rectificador** (M7) que evita que la corriente fluya en sentido contrario al deseado.

Además, Arduino UNO, cuenta con un **amplificador operacional** (identificado como "U5A" en los diagramas del circuito) que funciona en modo comparador de la siguiente forma: Si se conecta una fuente de alimentación externa a la placa (más exáctamente, si Vin > 6'6[V]) ésta, la utiliza, se activa la salida (que activar o desactiva el transistor "T1") del amplificador operacional y el transistor, que funciona a modo de interruptor, bloquea la alimentación que llega del USB **para que no se produzca conflicto entre las dos posibilidades de alimentación** de la placa; en caso contrario, el transistor permite que la alimentación de la placa provenga del cable USB conectado.

Finalmente, también en lo relacionado con la alimentación de la placa, existen, dentro de la zona etiquetada como "POWER", una serie de pines-hembra de alimentación del circuito y de referencia voltaica:



HW Arduino UNO, imagen 4

- **El pin Vin**: es un pin de **entrada de voltaje a la placa regulado a 5[V]**, cuando se está utilizando una fuente de alimentación eléctrica externa (con un voltaje, dentro de los márgenes de seguridad, de entre 7 y 12[V]); está conectado directamente a la entrada de alimentación para el conector jack de un adaptador AC/DC, por lo que este pin contendrá el mismo voltaje que con el que se está alimentando la placa. A través de este pin, **se puede suministrar tensión** (alimentación eléctrica) **a la placa** sin utilizar ni la entrada para jack ni la entrada para USB, sino **mediante una batería** (pila), **conectando el borne positivo de la batería a este pin** y el borne negativo al pin GND. De esta forma, la placa será alimentada directamente, y **el regulador de tensión** que incorpora **reducirá el voltaje recibido, al voltaje de trabajo de la placa**, 5[V]. Por otra parte, si la placa está siendo alimentada por una fuente externa, a este pin se puede conectar cualquier **componente** electrónico, **para alimentarlo directamente con el nivel de voltaje que aporta la fuente externa, sin regulación** por parte de la placa. Si la placa se alimenta mediante USB, este pin ofrecería un voltaje de 5[V], regulados por la placa. Aunque, en cualquier caso, la **intensidad de corriente máxima** aportada serían **40[mA]**.

- **Los pines GND**: pines conectados a tierra (es decir, a 0[V]). Es una referencia común que todos los componentes de los circuitos han de compartir. Existe otro pin GND cerca de los pines de entrada y salida digital de la placa. Se puede utilizar cualquiera de los pines GND, según nos resulte más cómodo uno u otro, a la hora de diseñar el cableado del circuito.

- El pin 5V: es el **pin de voltaje de salida de la placa de 5[V]** que pueden venir desde un adaptador AC/DC (7-12[V]), desde el pin Vin (7-12[V]) o a través del USB u otra fuente de 5[V] regulada. A través de este pin, se **puede suministrar tensión** (alimentación eléctrica) **a la placa** sin utilizar ni la entrada para jack ni la entrada para USB, sino mediante una batería (pila), **conectando su borne positivo a este pin** y su borne negativo al pin GND. **Y el regulador de tensión reducirá el voltaje recibido, a 5[V]**. Además, a este pin también se le puede conectar cualquier componente electrónico, para alimentarlo mediante una fuente externa, previamente regulada, con un voltaje de 5[V]. Y la **intensidad de corriente máxima** generada también es de **40[mA].**

- El pin 3'3V: es el **pin de salida de 3'3[V]** (con un margen de error de un 1%) a través de un circuito específico (el regulador de tensión "LP2985" incorporado en la placa) a partir del voltaje recibido indistintamente a través del conector para USB o el conector para el conector jack del adaptador AC/DC. La **intensidad de corriente máxima** generada, en este caso, será de **50 [mA**]. También se puede utilizar este pin para alimentar componentes del circuito (los más delicados que requieren dicho voltaje) pero, sin embargo, no se puede conectar una fuente externa a este pin, ya que el voltaje es demasiado limitado como para poder alimentar la placa. Aunque no se recomienda suministrar voltaje a la placa a través de los pines 5V y 3'3V porque no pasa por el regulador de tensión y se puede dañar la placa.

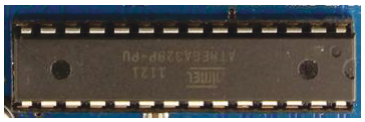
Junto a estos pines de alimentación, se encuentran otro dos pines E/S de la tarjeta Arduino de uso concreto:

- El pin **RESET** sirve simplemente para reiniciar la placa. Cuando el voltaje de este pin se establece a LOW, el microcontrolador se reinicia y se pone en marcha el *bootloader*. Para realizar esta función, la placa Arduino también consta de un pulsador *reset*, por lo que este pin puede ser de utilidad cuando se conectan sobre la placa Arduino *shields* o placas supletorias, que pueden ocultar e imposibilitar el uso del pulsador *reset* de la placa.

- El pin **IOREF** tiene como función indicar la **referencia de la tensión con la que está trabajando internamente la placa** (su microcontrolador), **5[V]** al tratarse de Arduino UNO. Puede ser de utilidad si se acoplan placas complementarias (*shields*) a Arduino UNO, y con este pin indicarlas el voltaje de trabajo de los pines E/S de Arduino UNO, para que se adapten automáticamente a dicho voltaje.

Existe, además de los anteriores, un pin extra que se encuentra sin etiquetar y actualmente no se utiliza. Está reservado para posibles usos futuros.

El microcontrolador


HW Arduino UNO, imagen 5

Un microcontrolador, comúnmente conocido como "micro", es un computador (parecido a un microprocesador, pero de mucha menor potencia, con funcionalidades de E/S y gestión de memoria, y pensado sólo para realizar tareas específicas) con prestaciones limitadas, en **un solo chip** (dispositivo electrónico que integra en un solo encapsulado un gran número de componentes).

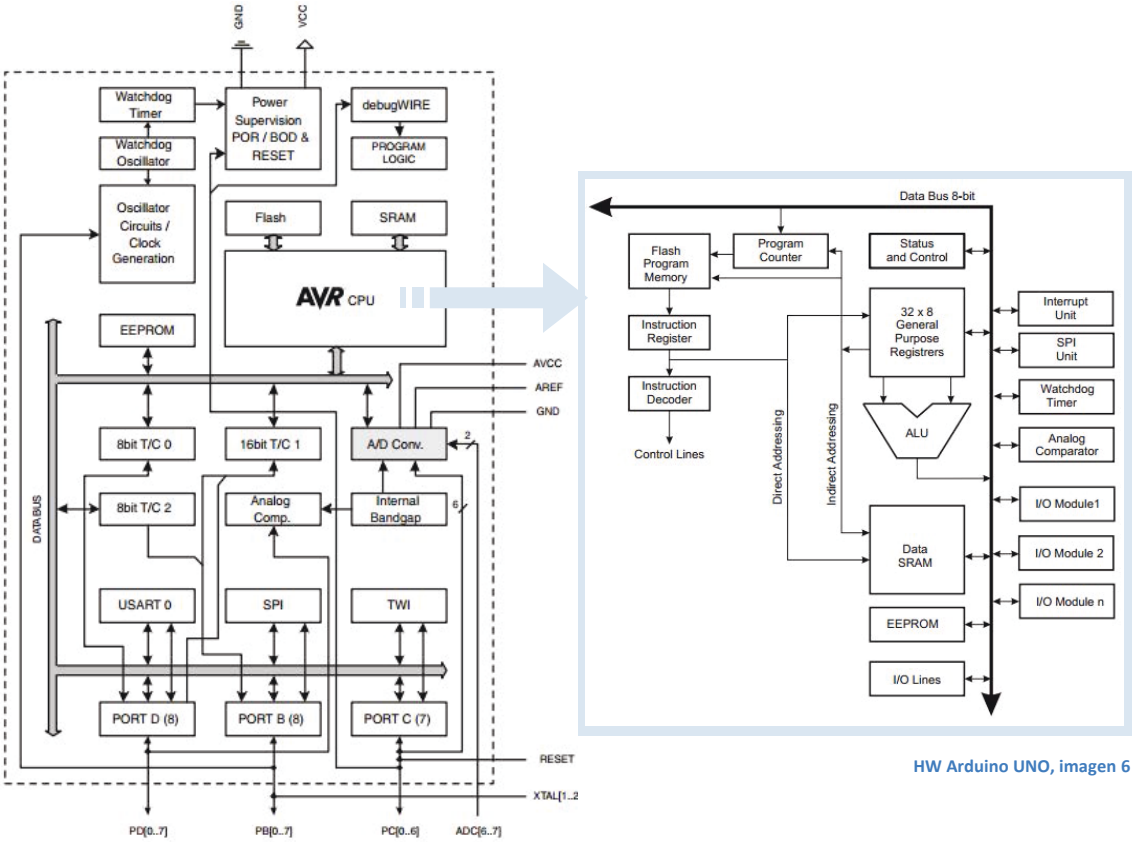
Se trata de **un circuito integrado** (con todos los elementos necesarios para funcionar, tales como, memoria, reloj oscilador y periféricos) **programable que puede ejecutar**, de forma **autónoma y constantemente, las instrucciones definidas** por el programador, teniendo en cuenta, en cada momento, la información obtenida y enviada a través de sus E/S, y reaccionando en consecuencia.

Los principales elementos que incluye el microcontrolador ATmega328P, son: CPU, memoria y pines E/S (conectores con forma de patillas). La CPU (*Control Process Unit*) ejecuta cada instrucción y controla que se realice correctamente (normalmente, las instrucciones utilizan los datos de entrada disponibles previamente, y generan datos de salida que serán utilizados, o no, por otra instrucción). Los **3 tipos distintos de memoria** del microcontrolador sirven para alojar las instrucciones y los datos que éstas necesitan (y que dicha información, instrucciones y datos, esté disponible para que la CPU trabaje con ella en el momento oportuno). De forma general, las memorias se suelen clasificar en **persistentes** (cuyo contenido se almacena de forma permanente, aunque se dejen de alimentar eléctricamente) o **volátiles** (cuyo contenido se pierde al dejar de recibir alimentación eléctrica) y según las características de la información a guardar, esta se grabará en un tipo un otro de memoria, habitualmente, de forma automática. El ATmega328P consta de una memoria volátil (Flash) y dos memorias persistentes (SRAM y EEPROM). Por su parte, los **23 pines E/S**, sirven para la comunicación del microcontrolador con los periféricos. Se dice que son pines GPIO (*General Purpose Input/Output*), porque son pines de propósito general, pero suelen tener otra determinada función específica. Entre estos pines se encuentra un pin **VCC** que recibe la alimentación eléctrica, un pin **AVCC** que recibe la alimentación suplementaria para el ADC (*Analog Digital Converter*) interno, **un pin AREF** que recibe la referencia voltaica analógica para el ADC y **dos pines GND** conectados a tierra.

El microcontrolador Amega328P incluye además, treinta y dos registros de propósito general, tres temporizadores (para gestión de interrupciones externas), protocolos de comunicación serie (UART, I^2C y SPI), un conversor analógico digital (ADC) de 10 bits y 6 canales, y un oscilador interno.

El microcontrolador es el cerebro de la placa. Es decir, el micro, es el componente principal de la circuitería de procesamiento y control de Arduino. Del *datasheet* (hoja de especificaciones)[41] de su fabricante se pueden extraer sus características y limitaciones. El microcontrolador de la placa Arduino UNO es el modelo **ATmega328P** de la marca **Atmel**. Es un microcontrolador CMOS de baja potencia que, mediante la ejecución de instrucciones en un solo ciclo de reloj, alcanza rendimientos en torno al millón de instrucciones por segundo (MIPS), por [MHz], lo cual permite optimizar el consumo de energía del sistema en el que se vaya a utilizar.

A continuación se muestra el diagrama de bloques del ATmega328P (a su izquierda, el diagrama de bloques de la arquitectura AVR).



HW Arduino UNO, imagen 6

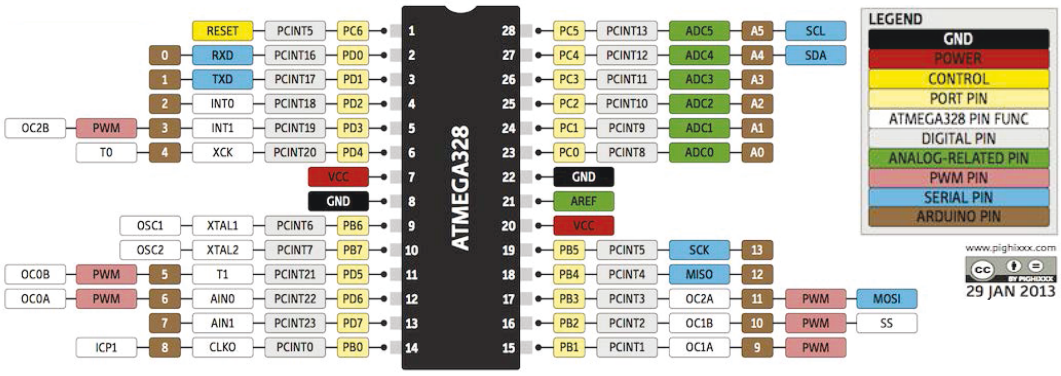
En general, la mayoría de las placas Arduino utilizan microcontroladores de la familia mega (ATmega). Arduino UNO, utiliza el modelo 328P; P significa que incorpora la tecnología Picopower (también de Atmel), para un consumo eléctrico menor que con el modelo ATmega328, es decir, que puede trabajar a menor voltaje y consumir menos corriente. En ocasiones, también se habla de este microcontrolador como ATmega328P-PU, donde PU se refiere al tipo de encapsulado del chip que, habitualmente, es de tipo DIP (patillas que atraviesan la placa donde se inserta a través de agujeros).

[41] El *datasheet* del ATmega328P se encuentra en: http://www.atmel.com/Images/doc8161.pdf.

Según la hoja de especificaciones del fabricante, puede funcionar en un rango de tensión de 1'8[V] a 5'5[V], y de temperatura entre -40[°C] a 85[°C].

En el caso de la tarjeta Arduino UNO, su microcontrolador ATmega328P puede encontrarse incorporado a ella **encapsulado** en formato DIP (*Dual In-line Package*) o SMD (*Surface Mount Device*). Si el micro se encuentra **en formato DIP**, estará conectado a la placa mediante las patillas metálicas (que funcionan como conectores o pines de E/S del microcontrolador) las cuales permiten separarlo de la placa si fuese necesario sustituirlo por otro. En otro caso, si el microcontrolador se encuentra **en formato SMD**, será más pequeño, para optimizar al máximo el espacio físico que ocupa, y estará soldado a la superficie de la placa mediante la tecnología SMT (*Surface Mount Technology*).

La imagen siguiente muestra el micro, con sus correspondientes conectores o pines de salida (y las funciones específicas que pueden llevar a cabo), encapsulado en el formato más común, DIP.



HW Arduino UNO, imagen 7

El ATmega328P, al igual que los microcontroladores de otros modelos de Arduino, pertenece a la **familia** de microcontroladores **AVR**[42] de Atmel, en concreto, a la subfamilia "**megaAVR**"[43]. La arquitectura del ATmega328P y en general, de toda la familia de microcontroladores AVR, es de tipo Hardvard modificada, de 8[bits], con tecnología RISC (*Reduced Instruction Set Computing*). Que tenga una arquitectura de tipo **Hardvard**[44] significa que la **memoria** que aloja los **datos** (la SRAM o EEPROM del micro de Arduino) se encuentra **separada de** la **memoria** que aloja las **instrucciones** (Flash del micro de Arduino). Ambas memorias (la de instrucciones y la de datos) se comunican con la CPU de forma independiente y en paralelo, lo que permite mayor velocidad y optimización de recursos.

[42] Todas las placas Arduino se basan en la arquitectura AVR (excepto Arduino Due, basado en la ARM). Además de AVR y ARM, existen otras familias de arquitecturas, como la PIC, desarrollada por Microchip, otro fabricante.
[43] Otras subfamilias de AVR podrían ser "tinyAVR" (de microcontroladores más limitados, llamados ATtiny) o "xmega" (de microcontroladores más complejos, llamados ATxmega).
[44] Otro tipo de arquitectura (la de los PCs, por ejemplo) es la Arquitectura Von Neumann, en la cual la CPU está conectada a una memoria RAM única que contiene tanto las instrucciones del programa, como los datos, por lo que la velocidad de operación está limitada (entre otras cosas) por el efecto de cuello de botella que se produce al haber un único canal de comunicación para datos e instrucciones).

El microcontrolador ATmega328P de Arduino UNO combina una memoria Flash interna de 32[KB] para almacenar el programa, una memoria SRAM de 2[KB] para datos volátiles y una memoria EEPROM de 1[KB] para los datos no volátiles.

- **Memoria Flash**. Se trata de una memoria persistente, donde se almacena de forma permanente (aunque se deje de alimentar eléctricamente a la placa) el **código del programa** (*sketch*) que ejecutará el microcontrolador, hasta una nueva reescritura, si la hubiese. En el ATmega328P tiene una capacidad de **32[KB]**, de los cuales, en Arduino UNO, 512[bytes] no se pueden utilizar porque están ocupados por el código del *bootloader* del microcontrolador. Si se adquiere un microcontrolador externo, separado de la placa Arduino, la capacidad esta memoria será de 32[KB] completos, puesto que el micro no tendrá el ***bootloader*** (necesario para que funcione al conectarlo a la placa Arduino) instalado. La memoria Flash, destinada en principio para almacenar el código del programa, también puede utilizarse para almacenar datos (variables), si no hubiese suficiente espacio en la SRAM para ello, o si se quisiese almacenarlos de forma permanente[45]. Los tipos de variables que se pueden mantener en la memoria Flash (y que no se podrán modificar, sino sólo leer, durante la ejecución del programa) suelen ser los de los tipos más habituales[46] que tienen unos nombres de tipos específicos. Pero la escritura en Flash provoca un desgaste en la memoria (sólo son posibles unos 10.000 ciclos de escritura, según el *datasheet* del microcontrolador). Aunque como en la mayor parte de las ocasiones no es necesario escribir en esta memoria, se suele reservar esta funcionalidad simplemente para las escrituras del *bootloader.*

- **SRAM** (*Static Random Access Memory*). Es un tipo de memoria volátil donde se alojan los **datos (variables, de un tipo concreto) que,** en ese instante, **el programa** (*sketch*), que se está ejecutando y que ha sido grabado separadamente en la memoria Flash, **necesita** crear, guardar o manipular **a lo largo de la ejecución** y cuyo valor se pierde al desconectar la alimentación eléctrica. La capacidad de esta memoria en el ATmega328P es de **2[KB]**. Si se necesita ampliar la cantidad de SRAM disponible, se pueden adquirir SRAMs independientes y conectarlas al microcontrolador mediante algún protocolo de comunicación (como SPI o I^2C). Decir además que el compilador maneja directamente la memoria SRAM, de forma transparente al programador[47].

[45] Para ello se utiliza el modificador de variables "`PROGEM`" o la función macro "`F()`".
[46] Concretamente, los que se encuentran definidos en el fichero de cabecera "avr\pgmspace.h".
[47] Al final del proceso de compilación, existe una parte (*linker*) que almacena las variables definidas por el programador en diferentes direcciones de esta memoria.

- EEPROM (*Electrically Erasable Programable Read Only Memory*). Se trata de una memoria persistente donde el programador puede almacenar los **datos que será necesario que permanezcan grabados una vez apagado** el microcontrolador, para poder utilizarlos posteriormente en siguientes reinicios. Es similar a la Flash, pero más económica y simple. Aunque más lenta porque no puede trabajar con bloques de datos. Pero es de gran utilidad, ya que permite, además de leer, escribir en ella, como también se puede en la SRAM. Para el ATmega328P la capacidad de esta memoria[48] es de **1[KB]** Y se puede utilizar a través de su librería estándar[49]. Además, si se necesita ampliar la capacidad de la memoria EEPROM, se pueden adquirir memorias EEPROM externas (o *shields* como las "USB Host") y conectarlas al microcontrolador a través de un protocolo (como SPI o I^2C); o también se puede adquirir un lector y memoria SD (o *shields* para SD que ya incorporan el lector), y manejarlas haciendo uso su librería estándar[50] y el protocolo de comunicación SPI.

Por otra parte, el microcontrolador ATmega328P de Arduino UNO cuenta con 32 registros de propósito general. Los **registros** del microcontrolador son espacios de memoria, de capacidad reducida (para almacenar unos pocos bits cada uno) dentro de la CPU del propio microcontrolador. Tienen varias funciones imprescindibles: **albergar los datos cargados** previamente desde la memoria SRAM o EEPROM, para tenerlos disponibles en el momento adecuado (que serán necesarios para la ejecución de próximas instrucciones); almacenar temporalmente los **resultados** de las instrucciones recientemente ejecutadas, por si se necesitan posteriormente; y alojar las propias **instrucciones que** en ese mismo momento **se estén ejecutando**. En un microcontrolador, cuanto mayor sea el número de bits que pueda almacenar cada registro, mayor serán sus prestaciones en cuanto a poder de cómputo y velocidad de ejecución. Cuando se habla de que un microcontrolador es de tantos bits, esto se refiere al tamaño de sus registros. El chip **ATmega328P es de 8 bits**, es decir, **los registros** del ATmega328P **son de 8 bits** (como ocurre en los micros de todas las placas Arduino, a excepción de la tarjeta Arduino Due, que es de 32 bits). Dependiendo del uso que se vaya a dar al microcontrolador, será necesario utilizar el que tenga un tamaño de registros suficiente.

Otro elemento importante del microcontrolador de Arduino UNO, es el *bootloader*. El *bootloader* (o gestor de arranque) del micro es imprescindible para un manejo de la placa cómodo y fácil ya que, **gestiona** (automática e internamente) **la grabación en la memoria Flash del programa**

[48] Que la capacidad de memoria sea de 1K, para hacernos una idea, quiere venir a decir que es como una matriz o tabla con 1024 posiciones, de 1 byte cada una.
[49] Para manejar la memoria EEPROM, a librería estándar de Arduino es "EPROM.h", que trabaja byte a byte. Se puede ver en el enlace: EEPROM. Si se desea, también se puede utilizar la librería extendida: "EPROMex", que puede consultarse en http://playground.arduino.cc/Code/EEPROMex.
[50] La librería estándar para lecturas y escrituras en una SD es: "SD.h". Enlace: SD.

que se va a ejecutar, permitiendo así programar la placa Arduino directamente por conexión USB entre el computador con el IDE y la placa. **Se trata de un programa, pregrabado de fábrica** dentro de la memoria Flash del microcontrolador (en un espacio reservado, en esa memoria, conocido como *bootloader block).* Este *software*, también denominado *firmware* porque no suele modificarse, es específico[51] según el tipo de microcontrolador. En **Arduino UNO**, el *bootloader* se llama Optiboot, ocupa **512 [Bytes] de la memoria Flash** y tiene una velocidad de grabación (de cada programa a cargar en el micro) de 115 [Kbits]/[seg].

El *bootloader* del micro **se ejecuta durante el primer segundo de cada reinicio** de la placa (su actividad se puede apreciar con el encendido de los leds TX y RX) en el momento de alimentar la placa). Durante esos instantes **espera a recibir, desde IDE,** el fichero con formato Intel Hex Format (extensión ***.hex**)[52] generado a partir del proceso de compilación del sketch (y con las instrucciones concretas e internas de programación para memorias Flash que, a pesar de que pueden ser ligeramente distintas según el tipo de *bootloader* de la placa, la mayoría son variantes del protocolo estándar STK500 que ofrece Atmel para la programación de sus microcontroladores) para interpretar sus **instrucciones** y cargar el programa, es decir, **para comenzar a grabar el programa.** Si transcurrido ese segundo no llegan dicho fichero con las instrucciones, el *bootloader* termina su ejecución y pasa la ejecución al programa principal, es decir, que comienza a procesarse el contenido que, en ese momento, tenga la memoria Flash. Sin embargo, **cuando le llegan las instrucciones precisas para que comience a grabar el programa** en la memoria, **recibe**, también **desde el IDE** (y normalmente, a través de un cable USB desde el computador donde se está ejecutando el IDE, hasta la placa) **el programa a ejecutar** en el microcontrolador **y lo almacena** correctamente **en la memoria Flash** del micro (en el espacio no ocupado ya por el propio *bootloader*) de manera automática y sin que el programador se tenga que preocupar del proceso.

Tras la grabación del programa en memoria, el *bootloader* termina su ejecución y el micro, inmediatamente (y de forma permanente, mientras continúe encendido), se dispone a procesar las instrucciones de dicho programa. **Intel Hex Format** es el formato que utilizan todos los chips AVR para almacenar contenido en sus memorias Flash, en la cual se almacenan internamente con este formato, tanto **los programas escritos en el IDE** como los ***bootloaders*** de los microcontroladores.

[51] El *bootloader* de cada placa viene determinado, según de qué microcontrolador se trate, a partir de la información del fichero "boards.txt" descargado con el IDE de Arduino. Es habitual que derive del protocolo estándar STK500, cuyos detalles puede consultarse en: http://www.atmel.com/Images/doc2525.pdf.

[52] El fichero Intel Hex contiene líneas de texto con las instrucciones para la memoria Flash. Cada línea de texto contiene los datos sobre el número de bytes de línea, la dirección de comienzo de los datos en la Flash que les siguen, un *flag*, el código máquina que se debe programar y un código de comprobación de errores.

Si se adquiere un microcontrolador ATmega328P independiente de la placa Arduino, éste no vendrá con el *bootloader* instalado de fábrica y habrá que, o bien incorporarlo, o bien cargar los programas directamente a la memoria Flash, para no tener que utilizar *bootloader*. En cualquier caso, será necesario adquirir un ISP (*In-System Programmer*), un hardware programador específico, que se ha de conectar al computador y a la placa Arduino, para que (por ausencia del *bootloader*) actúe de intermediario entre el IDE y la memoria Flash del micro. Además, los *bootloaders* de Arduino también son *software* libre; al igual que ocurre con el IDE Arduino, disponemos de su código fuente (en C) para conocer cómo funciona internamente e incluso modificarlo, si se considerase oportuno.[53]

El reloj oscilador

HW Arduino UNO, imagen 8

El reloj de Arduino UNO es un **resonador** (a veces también llamado oscilador) que funciona **a una frecuencia de 16[MHz]**, que **sirve al micro para controlar** la velocidad a la cual va ejecutando las instrucciones del código máquina (así como la velocidad de lectura y escritura de los datos en su(s) memoria(s), la velocidad de adquisición de datos en los pines de entrada, y la velocidad de envío de datos hacia los pines de salida) determinando, por lo tanto, **su frecuencia** (o ritmo) **de trabajo**. Una frecuencia de reloj de 16[MHz] quiere decir que el microcontrolador, aproximadamente, es capaz de realizar 16 millones de instrucciones por segundo, es decir que, una instrucción máquina se ejecutará cada (1*1/16.000.000 = 0'0000000625 segundos) 62'5[ns]. Y a pesar de que sería posible que la placa tuviese un reloj con una capacidad de trabajo mayor (para disminuir el tiempo en que se ejecutan las instrucciones) esto implicaría un incremento en el consumo de energía y en el calor generado.

El reloj que incorpora la placa **Arduino UNO** es un **resonador cerámico.** En el ámbito electrónico, existen varios tipos de relojes; entre ellos se encuentran los osciladores (también llamados resonadores) de cristal, y los cerámicos. Los osciladores de cristal son circuitos que utilizan un material piezoeléctrico de cristal (generalmente de cristal de cuarzo) para generar una onda vibratoria de alta frecuencia muy precisa. Mientras que, los resonadores cerámicos, están formados por un material piezoeléctrico cerámico (como puede ser el PZT o zirconato titanato de plomo), que genera la señal oscilatoria de la frecuencia deseada, al aplicársele un determinado voltaje. Los osciladores cerámicos son ligeramente menos precisos que los de cristal (exactamente tienen un 0'5% de precisión, respecto al 0'001% de los de cristal de cuarzo), pero más económicos.

[53] El código fuente (copias exactas bit a bit) de los *bootloaders* oficiales grabados en los micros de Arduino se encuentra en el paquete instalador del IDE de Arduino (carpeta "hardware\arduino\bootloaders"). Son ficheros Intel Hex Format con extensión "*.hex" que, si hubiese que grabar el *bootloader* de un microcontrolador (y disponiendo de un programador hardware ISP) se pueden cargar en la memoria Flash del microcontrolador.

Si no estamos satisfechos con la precisión del reloj, **se puede utilizar un componente externo** que actúe como tal y sincronizar el microcontrolador con él. Técnicamente, también se podría utilizar, como reloj del ATmega328P, **el oscilador de cristal** que controla el **chip ATmega16U2** pero, a efectos prácticos, el circuito resultante generaría excesivo ruido electromagnético. Por otro parte, el microcontrolador **ATmega328P** también incluye un **reloj interno propio** (de tipo RC) dentro de su encapsulado, por lo que, en teoría, no sería necesario utilizar ningún reloj adicional. Aunque como ese reloj interno funciona a una **frecuencia de 8[MHz]** y tiene una **baja precisión** (de un 10%), no suele utilizarse.

El microcontrolador tiene **diferentes opciones de origen del reloj**[54] seleccionables a través de un registro interno del microcontrolador. La señal de reloj sale del multiplexor (*Clock Multiplexer*), pasa a través de un sistema de prescalado (*System Clock Presacaler*) para reducir la frecuencia y el consumo y mejorar la estabilidad de la señal, y **finalmente** llega a la unidad de control (*AVR Clock Control Unit*[55]) que distribuye y gestiona el **reloj para los distintos periféricos** del microcontrolador.

Conversión USB a serie (chip ATmega16U)

HW Arduino UNO, imagen 9

La conexión USB de la placa Arduino UNO, además de para la alimentación eléctrica de la placa, también sirve como puerto (o interfaz de comunicación) serie, para **transmitir información entre el computador y la placa** a través del **protocolo USB**, que el computador y la placa son capaces de manejar. **El ATmega328P**, por sí mismo, **no entiende el protocolo USB** (sino otros más sencillos como el I^2C o el SPI). Para manejar la información transferida a través del USB, la placa Arduino UNO R3 dispone de **otro microcontrolador**, el **ATmega16U2**, con su propia CPU y su propia memoria (Flash de 16[KB]).

El ATmega16U2 está **programado**[56] de fábrica **con un** *firmware,* para que pueda actuar como **intérprete** (es decir, traductor o conversor) **de USB a serie** permitiendo así, la comunicación entre el

[54] Las opciones de reloj son: un oscilador de cristal de bajo consumo, uno de cristal a pleno funcionamiento, uno de cristal a baja frecuencia, uno interno RC a 128[kHz], uno interno RC calibrado, o uno externo.

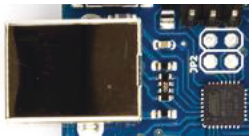
[55] La *AVR Clock Control Unit* distribuye y gestiona el reloj para los distintos periféricos del micro, dividiendo los relojes en: **clk_{CPU}**, para la **CPU y la RAM**; **clk_{FLASH}**, para los pulsos de reloj que controlan las operaciones de la **Flash y la EEPROM** (se suele activar conjuntamente con el clk_{CPU}); **clk_{ADC}**, para el **conversor ADC** (al separar esta señal de reloj del resto se reduce el ruido generado por la circuitería digital y la conversión analógica a digital es más exacta); **$clk_{I/O}$**, para la mayoría de **módulos E/S** (como contadores/*timers*, el SPI y el USART); y **clk_{ASY}, para el contador/*timer* asíncrono** (el timer puede utilizar un cristal de cuarzo de 32[kHz] para obtener una señal a partir de un reloj externo y así poder utilizar el *timer* como reloj en tiempo real, que sigue funcionando incluso cuando se está en modo dormido o de bajo consumo).

[56] El ATmega16U2 de la placa ya viene preprogramado con un *software* libre que, si se desea consultar, se encuentra disponible en un fichero "*.hex", en la carpeta de Arduino, en "hardware\arduino\firmwares".

computador y el microcontrolador ATmega328P, a través del USB[57] y del dispositivo receptor y transmisor para comunicaciones serie, conocido como USART (*Universal Synchronus or Asynchronus Receiver and Transmiter*). El micro ATmega16U2 se inicia automáticamente con el gestor de arranque DFU que incorpora de fábrica. Incorpora también, un **reloj** oscilador de cristal para mantener la sincronización con la comunicación USB. Además, consta de su propio **conector ISCP** (*In-Serial Circuit Programming*) en caso de que se quiera desprogramarlo y programarlo de nuevo, para actualizarlo o para que, en vez de funcionar como un simple traductor de USB a serie, pueda simular cualquier otro tipo de dispositivo USB conectado al computador (como un ratón, un teclado, o un dispositivo MIDI, por ejemplo). [58]

En las versiones de placas Arduino USB anteriores a la placa Arduino UNO, en vez del micro ATmega16U2 (ATmega8U2, hasta la versión UNO R3) existía otro circuito intérprete de USB a serie, concretamente el FT232RL, del fabricante FTDI (*Future Technology Devices International*), que era menos económico y no tenía la posibilidad de ser reprogramado.

Comunicación serie



HW Arduino UNO, imagen 10

En una comunicación de tipo serie, la información se transmite bit a bit (uno tras otro) teóricamente **por un único canal de transmisión**. El envío de los bits con los datos viene, precedido del envío de un bit de comienzo, y seguido del envío de un bit de parada. La principal diferencia respecto a una comunicación en paralelo es que, en paralelo, se necesitarían varios canales y se enviarían varios bits simultáneamente (cada uno, por un canal de transmisión separado y sincronizado con el resto), con el correspondiente incremento de complejidad, tamaño y coste del circuito resultante que todo ello supondría.

En la comunicación serie intervienen tanto el *software* como el *hardware*. El ***software*** tiene el control de **qué es lo que se va a enviar (qué bits) y se lo indica al *hardware*** mediante ciertas llamadas (que en el caso de Arduino, son librerías que evitan tener que trabajar con el *hardware*

[57] El ATmega16U2 permite canalizar la comunicación serie a través del USB, y que aparezca como un puerto COM virtual ante el software instalado en el ordenador. Utiliza los controladores COM USB estándar, por lo que no es necesario utilizar *drivers* externos (sólo es necesario configurarlo en Windows, como se explica en el apartado de "Instalación del IDE" de este documento, o en el sitio web de Arduino (Enlace: Install the drivers).

[58] En caso de querer modificar el firmware del Atmega16U2, se puede utilizar el software de Atmel FLIP (para Windows) o el programador DFU (para Mac OSX y Linux). Para obtener más información se puede consultar un tutorial aportado por los usuarios de Arduino (Enlace: tutorial)

directamente). El **hardware**, por su parte, controla **cómo lo va a enviar**, enviando y recibiendo **pulsos eléctricos** que conforman los datos transmitidos.

Las comunicaciones en la placa Arduino UNO pueden establecerse, tanto entre la placa y el computador, como con otras placas Arduino, o con otros microcontroladores. En la **transmisión de información vía USB** entre el computador y la placa (es decir, entre el computador y el ATmega 328P, que a su vez se comunica con el ATmega16U2, que actúa de intermediario entre la placa y el USB) se utiliza el **puerto de comunicaciones serie UART**[59].

Los **pines de conexión** de Arduino UNO relacionados con la comunicación serie son: **el pin 0, RX** (para recepción) **y el pin 1, TX** (para transmisión); y, sólo para el modo síncrono, el pin 13, SCK (reloj). Estos pines permiten establecer una comunicación serie entre los dispositivos externos y chip UART-TTL, sin necesidad de intermediarios. Ambos pines están **conectados internamente al chip ATmega16U2** (a través de unas resistencias internas de 1[KΩ]); así los datos que se encuentren disponibles en el USB también lo estarán en estos pines. Pero cuando se utilice la transmisión serie, los pines 0 y 1 de la placa estarán ocupados y no podrán ser utilizados como pines de entrada y salida digital; los leds de la placa marcados como "TX" y "RX", parpadean para indicar actividad en el puerto serie.

El chip UART que incorpora la placa, disponible en los pines digitales 0(RX) y 1(TX), ha sido construido con la tecnología **TTL** (*Transistor to Transistor Logic*). Arduino utiliza los niveles TTL (de lógica de transistor a transistor, donde los elementos E/S son transistores bipolares)[60] en los cuales **el 0** lógico (es decir, que cuando el bit es un 0), **o** valor **LOW del pulso**, se **corresponde** a **0[V]** y **el 1** lógico **o valor HIGH** del pulso, se corresponde a **5[V]** (ó 3'3[V] si fuese ese el voltaje de trabajo del circuito). Este hardware UART integrado en la placa, permite al microcontrolador recibir datos en serie (mientras haya espacio en el *buffer* de 60 bytes), aunque esté trabajando en otras tareas.

El **manejo del UART**, como ocurre con el resto de periféricos del micro, se lleva a cabo mediante una serie de registros, que se encuentran mapeados en memoria (de forma que cuando se accede a una determinada dirección de memoria, en realidad se está accediendo a un registro). Estos registros permiten configurar el funcionamiento del UART y realizar las lecturas y escrituras de los datos que se van enviar y recibir. El UART puede configurarse en lo relativo al modo de trabajo (síncrono, *UART*, o asíncrono, *USART*), a la velocidad de transmisión, al reloj que se utilice (interno o

[59] Para mayor información sobre la comunicación serie con UART se puede consultar el siguiente enlace a la Wikipedia: http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter.

[60] Arduino utiliza los niveles TTL, donde un bit 0 (LOW) corresponde a 0[V] y un bit 1 (HIGH) corresponde a 5[V] (o 3'3[V] si fuese esa la tensión de funcionamiento y trabajo de la placa), frente a otros niveles, como podrían ser, por ejemplo, los niveles DDL (*Diode to Transistor Logic*) donde el 0 lógico es 1[V].

externo), a la cantidad de bits de datos que se van a transmitir (entre 5 y 9), al bit de paridad (si lo tiene y es par o impar, o si no lo tiene), o los bits de parada (1 ó 2). Cuando está configurado y se escribe un registro el dato, el hardware UART lo serializa (añade los bits de inicio y parada y calcula el de paridad) y lo envía en el orden adecuado; para recibirlo (sirviéndose del *buffer*) hace el proceso inverso, dejando el dato capturado en un registro, preparado para ser leído. **Arduino**, para abstraer al programador de estos detalles y evitar tener que acceder directamente a la memoria, **proporciona una serie de librerías y funciones[61].** Además, el IDE de Arduino incluye un monitor serie[62] que permite enviar y recibir desde la placa, datos textuales sencillos.

Normalmente, las placas Arduino tienen un puerto serie UART, con el conversor USB a serie integrado (si no lo incorporase sería necesario un adaptador de TTL a USB), que se corresponde a los pines 0 y 1 de la placa. Pero también se pueden encontrar placas con varios puertos serie[63], e incluso, mediante código, el programador puede, haciendo uso de librerías y funciones[64], habilitar otros pines para que trabajen como puerto serie.

Existen diversos protocolos y estándares basados en la transferencia de información serie, que implementan de forma diferente cada uno de los detalles específicos de la comunicación (como el modo de sincronización entre emisor y receptor, la velocidad de transmisión, el tamaño de los paquetes de datos, los mensajes de conexión y desconexión y de paso al otro en el intercambio de información, o los voltajes recibidos, entre otras). **De entre** todos **los protocolos de comunicación serie** estándar reconocidos por la gran variedad de dispositivos electrónicos del mercado, **el ATmega328P,** es capaz de comprender y utilizar para contactar con dicha variedad de periféricos (y así, comunicar la placa con dispositivos externos), principalmente dos: **el I^2C** (o TWI)[65] **y el SPI[66]**.

El protocolo **I^2C** (*Inter-Integrated Circuit*) también denominado TWI (*Two-wire,* dos cables), para la transmisión de información serie, se basa en utilizar **dos líneas** o hilos **bidireccionales de bus**

[61] Todas las acciones referentes a la comunicación serie se realizan con la clase "`Serial`" de Arduino (se puede consultar en el enlace Serial). Antes de iniciar la comunicación a través del puerto serie, hay que configurar la función "`Serial.begin();`" indicando como parámetro la velocidad (medida en baudios por segundo, [bps]) de transmisión de información que se desea, durante la comunicación. La comunicación serie se puede finalizar en cualquier momento con "`Serial.end();`".

[62] El monitor serie que aparece en el IDE Arduino, se explica de forma más extensa en el apartado "Ejecución del IDE" de este documento.

[63] Las placas Arduino de la gama Mega, incorporan varios puertos serie. Aunque, como el conversor USB a serie integrado sólo se encuentra en los pines 0 y 1, para utilizar los demás puertos se necesitarían adaptadores.

[64] Para poder utilizar la comunicación serie a través de cualquiera de los pines de Arduino, se recomienda utilizar la librería "`SoftwareSerial.h`" de Arduino (enlace SoftwareSerial).

[65] La comunicación serie de la placa con dispositivos externos mediante el protocolo I^2C se puede gestionar con la librería "`Wire.h`" de Arduino (enlace Wire).

[66] La comunicación serie de la placa con dispositivos externos mediante el protocolo SPI se puede gestionar con la librería "`SPI.h`" de Arduino (enlace SPI).

(además de las otras dos necesarias, una para alimentación y otra para tierra común, que se presuponen existentes en el circuito). **Estas dos líneas son:**

- Línea **SDA** (*data line*), la **única para** transferir los **datos** (en forma de 0s y 1s). En el ATmega328P se encuentra en su pin 27, que corresponde al pin A4 de la placa Arduino. Al haber una única línea de datos, la transmisión de información es "*half duplex*", es decir, **la comunicación sólo se puede establecer en un sentido al mismo tiempo**, por lo que, en el momento en que un dispositivo empieza a recibir un mensaje, para poder responder, tiene que esperar a que el emisor deje de transmitir.

- Línea **SCL** (*Serial Clock Line*) para enviar la señal de reloj. En el ATmega328P se encuentra en su pin 28, que corresponde al pin A5 de la placa Arduino. La señal de reloj es una señal binaria de una frecuencia periódica muy precisa, que coordina y **sincroniza** a los emisores y receptores en la comunicación para que sepan cuándo comienza la comunicación, cuánto dura y cuándo termina la transferencia de información.

Para que la comunicación serie se realice correctamente, tanto la línea **SDA** como la línea **SCL** han de estar **conectadas a la fuente de alimentación** común, **mediante una resistencia "*pull-up*" por cada línea**, que algunos chips ya tiene integrada. **Cada dispositivo que** se conecta al bus para participar en la comunicación serie (hasta un máximo de 128 dispositivos) tiene una **dirección única** que lo diferencia del resto, **y puede estar configurado como maestro** (*master*) **o esclavo** (*slave*). El maestro (que no tiene por qué ser siempre el mismo dispositivo, si no que los distintos dispositivos se pueden intercambiar ordenadamente esta función) inicia la transmisión de datos y genera la señal de reloj; y el esclavo, se limita a responder. La velocidad de transferencia de información que ofrece I^2C es de 100 kilobytes por segundo (o lo que es lo mismo, 100[Kbps])en el modo estándar, aunque se permiten velocidades hasta unos 3'4[Mbps]. Se utiliza con frecuencia en la industria para comunicar circuitos integrados entre sí. Debido a su extenso uso, la mayoría de microcontroladores incluyen un *hardware* (periférico) que se encarga de toda la gestión del bus.

El protocolo **SPI** (*Serial Peripheral Interface*) es algo diferente. También proporciona un interfaz para la comunicación serie síncrona en distancias cortas, del microcontrolador, con dispositivos electrónicos digitales (periféricos u otros microcontroladores). Además, al igual que con el I^2C, un dispositivo conectado al bus SPI puede ser maestro (el microcontrolador, normalmente) o esclavo (el periférico, normalmente). Pero el SPI **requiere cuatro líneas** (cables), en vez de dos:

- Línea **SCK** (*Serial Clock*), envía a todos los dispositivos la señal de reloj que genera el dispositivo que sea el maestro en ese momento. El reloj sincroniza así, mediante sus pulsos,

las transmisiones del maestro. En el ATmega328P esta línea se encuentra en su pin 19, que corresponde al pin 13 de la placa Arduino.

- Línea **SS** (*Slave Select*). Es la que utiliza el maestro para seleccionar con qué esclavo se va comunicar, de entre los varios que puede haber conectados, ya que sólo puede transferir datos con uno a la vez. **El esclavo que reciba** por su línea SS un valor de voltaje a nivel **LOW, será el que esté en ese momento seleccionado** por el maestro, y los que reciban un valor de voltaje a nivel HIGH (el resto de pines) no lo estarán. Cuando hay varios esclavos conectados al circuito, es necesario utilizar una línea SS diferente por cada uno de ellos, para lo cual se puede utilizar cualquier otro pin digital; esto no ocurre con las tres líneas restantes, SCK, MOSI y MISO, compartidas por todos los dispositivos.

- Línea **MOSI** (*Master Out, Slave In*) para enviar los datos (0s y 1s) desde el maestro hacia el esclavo seleccionado. En el ATmega328P, corresponde se encuentra en su pin 17, que corresponde al pin 11 de la placa Arduino.

- Línea **MISO** (*Master In, Slave Out*) para enviar la respuesta (0s y 1s) del esclavo al maestro. En el ATmega328P, se encuentra en su pin 18, unido al pin 12 de la placa Arduino. Al haber dos líneas para los datos, la transmisión de información es "*full-duplex*", porque puede ser enviada en los dos sentidos (desde el maestro al esclavo, y viceversa) a la vez.

El protocolo **SPI,** respecto al I^2C, tiene la ventaja de que **es más rápido y consume menos energía; aunque** la principal desventaja es que **exige** que el microcontrolador tenga que dedicar muchos **más pines para la comunicación externa**. Normalmente los microcontroladores también incluyen un *hardware* específico para la gestión del SPI[67].

Conector de pines E/S de la placa

Los pines E/S de la placa Arduino sirven para que la placa se pueda comunicar con el entorno y que podamos interactuar con ella. La mayoría de estos pines (sin tener en cuenta los de la zona de alimentación) se encuentran internamente unidos a pines del microcontrolador. En los pines de **entrada** se pueden conectar **sensores** para que la placa (su microcontrolador) reciba datos desde el entorno, mientras que en los de **salida**, es posible conectar actuadores a los que la placa (su micro) pueda enviar órdenes e interactuar así, con el entorno que nos rodea.

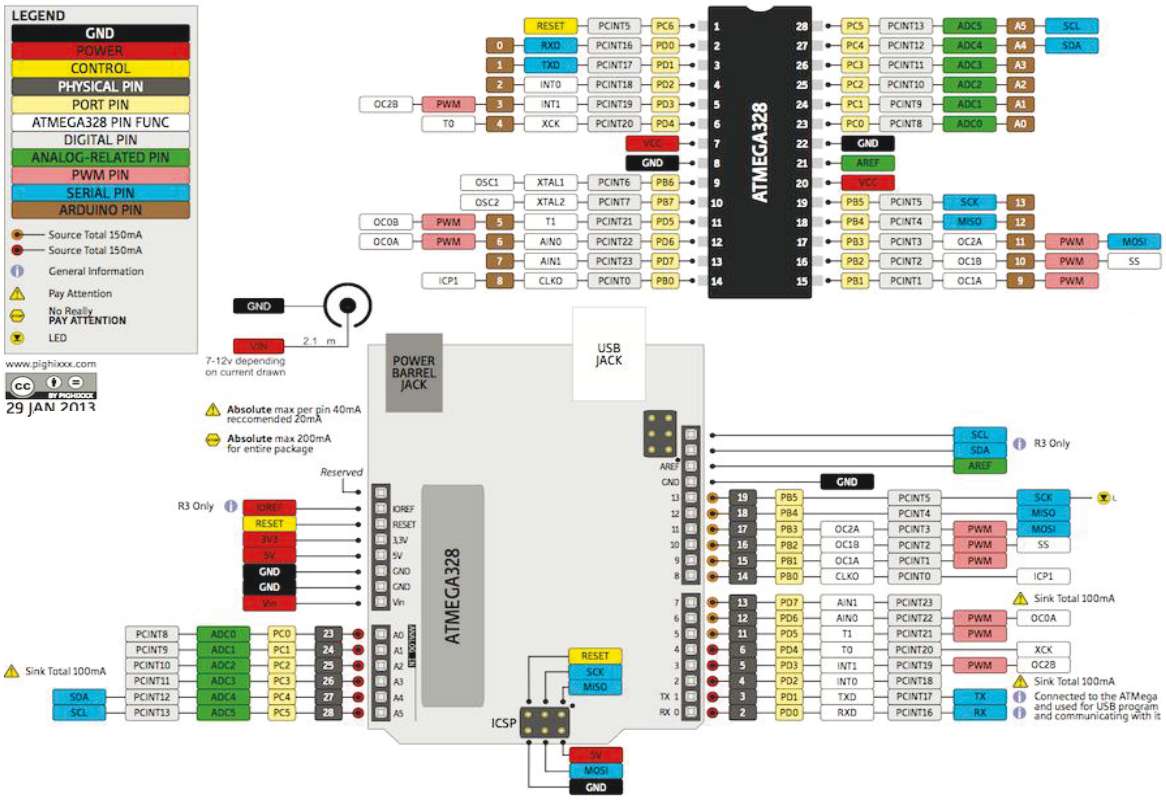
[67] Para la comunicación según el protocolo SPI, es conveniente utilizar la librería SPI (Enlace: SPI). También se puede utilizar el USART como SPI, ya que es compatible con la configuración de maestro SPI, aunque en la práctica, lo habitual es la configuración compatible con los puertos serie RS232 habituales.

Además de los **8 pines de la zona de alimentación** (el inutilizado, el pin **IOREF**, el pin **RESET**, el pin **3'3V**, el pin **5V**, los dos pines **GND**, y el pin **Vin**), que se explicaron en un subapartado anterior ("La alimentación eléctrica"), la placa cuenta con **14 pines de entrada o salida digital** (6 de los cuales también se pueden configurar como salidas analógica), **6 pines de entrada analógica y otros 4 pines** específicos, al lado de los 14 pines de E/S digital (**GND**, **AREF**, **SDA** y **SCL**).

Cada uno de los 14 pines E/S digital, como su propio nombre indica, se puede utilizar como entrada o como salida digital, utilizando las funciones de programación correspondientes. Estos pines operan a una tensión de 5[V]. Cada pin puede proporcionar o recibir un máximo de 40[mA] y tiene una resistencia *pull-up*, es decir, una resistencia conectada a +5[V] (desconectada por defecto) de 20 a 50[KΩ]. Además, algunos de estos pines digitales también tienen funciones específicas. Se explican más adelante.

Por su parte, los 6 pines de entradas analógica proporcionan 10 bits de resolución (es decir, la posibilidad de trabajar con 1024 valores diferentes, de 0 a 1023), trabajan, por defecto, a 5[V] y, algunos de ellos, también cuentan con funciones especializadas. Se explican más tarde. Finalmente, los 4 pines restantes de la placa, GND, AREF, SDA y SCL, se explican más adelante.

En esta imagen se muestran los pines de entrada y salida de la placa Arduino UNO R3 y, en la parte superior izquierda, de su microcontrolador.



HW Arduino UNO, imagen 11

Entradas y salidas digitales de Arduino UNO

HW Arduino UNO, imagen 12

La placa Arduino dispone de 14 pines de tipo hembra, que funcionan como entrada o salida digital[68] y se encuentran numerados del 0 al 13 en la zona de la placa etiquetada como "DIGITAL (PWM~)". A veces se habla de estos pines como pines GPIO (*General Purpose Input/Output*). A estos pines **se pueden conectar componentes, como sensores** (para recibir datos del entorno a través de la placa) **o actuadores** (para enviar órdenes a los sensores a través de la placa) **externos**. Pueden funcionar, por tanto, como salida o entrada digital, y 6 de ellos (los PWM~), se pueden configurar como salidas analógicas.

Estos 14 pines de E/S digital **funcionan a 5[V]** y la intensidad máxima que pueden recibir o proporcionar es de 40[mA] (aunque internamente, la placa agrupa estos pines de tal forma que, en conjunto, los pines 0 a 4, 100[mA] a la vez, y los pines 5 a 13, sólo pueden aportar 100[mA] a la vez; así que, como mucho, sólo se podrán tener 10 de estos pines ofreciendo 20[mA] a la vez). Constan de una resistencia "*pull-up*" interna de entre 20 y 50[KΩ] (que por defecto se encuentra desconectada y, el programador, puede activar).

Determinados pines digitales de la placa, además de la función estándar de entrada y salida digital, pueden realizar otras funciones específicas. El **pin 0 (RX)** sirve para que el microcontrolador ATmega328 pueda **recibir datos** en serie y el **pin 1 (TX)**, para que pueda **enviar datos** en serie, todo ello sin tener que realizar la conversión de USB a serie, porque ya se encuentran conectados a los pines del ATmega16U. Los **pines 3**, **5**, **6**, **9**, **10** y **11**, con la serigrafía "PWM ~", se pueden utilizar para simular salidas analógicas de Arduino (de 8 bits), que se explican más adelante. Los **pines 10** (SS), **11** (MOSI), **12** (MISO) y **13** (SCK) soportan la comunicación SPI, para la conexión de dispositivos con los que se vaya a establecer una comunicación a través este protocolo[69]. Y los **pines 2 y 3** se pueden programar para la gestión de interrupciones externas[70]. El **pin 13**, por su parte, se encuentra conectado internamente al led de la placa (marcado como "L"), es decir que, si este pin recibe un valor de voltaje alto (HIGH), correspondiente a 5[V], el led se enciende; mientras que si recibe un voltaje bajo (LOW), correspondiente a 0[V], el led se apaga; se trata de una forma sencilla y cómoda de detección de entradas externas, sin utilizar un componente externo.

Situados **junto a la zona de** la placa etiquetada como "DIGITAL (PWM~)" existen **unos pines de uso específico**, además de **otro pin GND**.

[68] Las principales funciones para tratar los pines digitales son: `pinMode()`, `digitalWrite()` y `digitalRead()`.
[69] La comunicación serie bajo el protocolo SPI se puede gestionar con la librería SPI. Enlace: librería SPI.
[70] Para la gestión de interrupciones se puede utilizar la función `attachInterrupt()`. Enlace: attachInterrupt ().

- El **pin AEREF** ofrece un voltaje de referencia externo para aumentar la precisión de las entradas analógicas. Este pin, se explica un poco más adelante (en "Entradas analógicas").

- El **pin SDA** y el **pin SCL** ofrecen la posibilidad de conectar algún dispositivo con el que se quiera establecer una comunicación bajo el protocolo I^2C (o TWI).

Por otra parte, para ampliar el número de pines de entrada o salida digital disponibles, los pines de entrada analógica **también pueden utilizarse como pines de entrada o salida digital**, considerándolos como pin 14 (en el caso del pin A0), pin 15 (el pin A1), … ó 19 (A5).

Entradas analógicas[71] de Arduino UNO

HW Arduino UNO, imagen 13

La tarjeta Arduino UNO está formada por 6 entradas **analógicas**. Son los pines de tipo hembra numerados de "A0" a "A5" en la zona de la placa etiquetada como "ANALOG IN". **Pueden recibir valores continuos de voltaje, entre los 0 y los 5[V], pero** como la placa sólo puede trabajar con valores digitales, **es necesaria una conversión previa** del valor analógico que reciba, **a un valor digital lo más aproximado posible**. Esta conversión se realiza mediante un conversor analógico/digital (ADC, *Analog-to-Digital*) con 6 canales (uno por entrada) y **10 bits de resolución en los que se almacena el valor de voltaje digitalizado**.

El número de bits de resolución, es el número de bits que tiene un ADC reservados para almacenar el valor digital de un valor de voltaje analógico. Influye en el grado de precisión del ADC, puesto que cuanta mayor cantidad de bits de resolución tenga, más fidedigna será la transformación al valor digital. Tener 10 bits de resolución quiere decir que **puede trabajar** internamente con 2^{10}= **1024 valores digitales** distintos (de 0 a 1023). El voltaje analógico mínimo que tiene que haber entre dos valores analógicos, para que el ADC los trate como valores digitales distintos se obtiene dividiendo el rango de los posibles valores de entradas analógicas entre el número máximo de valores digitales que puede tomar el ADC, es decir, ((5[V] – 0[V]) / 1024 = 0'0048[V]) 5[mV]. Para mejorar la precisión del ADC, como no se puede incrementar el número de bits de resolución, es posible reducir el voltaje de referencia (o límite superior del rango de posibles valores analógicos de entrada) que son 5[V].

Mediante las entradas analógicas se puede leer un rango de valores (analógicos digitalizados) que **habrá que** interpretar, para **transformar el valor leído en un valor útil**. Ya que Arduino UNO

[71] Si se desea consultar el tutorial sobre las entradas analógicas que se presenta en el sitio web de Arduino, se puede acceder a través del siguiente enlace: http://arduino.cc/en/Tutorial/AnalogInputPins.

puede trabajar con valores digitales de 0 a 1023, para saber el valor real que representa un valor en esa escala, ha de transformarse a transformarse a una escala real[72].

Al igual que los pines de E/S digital, algunos pines de entrada analógica tienen, **además de su función** de entrada analógica, o entrada o salida digital, **otra función específica**. En concreto, el pin **A4** y el pin **A5**, pueden ser utilizados para la comunicación bajo el protocolo I^2C[73], como **líneas SDA y SCL, respectivamente**.

Por otra parte, el pin **AREF** (*Analogue REFerence*) sirve para indicar a la placa que utilice otra referencia voltaica de la señal a medir, un valor de voltaje constante entre 0 y 5[V], que en la tarjeta Arduino UNO se puede seleccionar a través de una función del lenguaje de programación Arduino, específica para ello[74]. No funciona como entrada ni como salida, puesto que tiene el uso concreto de ofrecer un voltaje de referencia externo para poder aumentar la precisión de las entradas analógicas.

Como ya se ha comentado anteriormente, los pines de entrada analógica **también pueden utilizarse como pines de entrada o salida digital**, numerándoles como pin 14 (correspondiente al pin A0), pin 15 (el A1), … ó 19 (el A5).

Salidas analógicas[75] de Arduino UNO

La placa Arduino UNO **no dispone** exactamente **de pines de salida analógica en sí**, como para representar señales las analógicas que van tomando distintos valores continuos en el tiempo (variando gradualmente entre su mínimo y su máximo, es decir, tomando valores intermedios). Si no que **para simular un comportamiento analógico**[76], mediante salidas digitales en forma de pulsos, **se sirve de** algunos **pines de salida digitales** etiquetados como "PWM~".

[72] Para la transformación de un valor analógico digitalizado a un valor de entre 0 y 1023, a su valor real, se puede utilizar la función "`map(valor, minOrig, maxOrig, minDest, maxDest);`" que transforma `minOrig` en `minDest` y `maxOrig` en `maxDest` e interpola los valores intermedios, devolviendo el valor transformado (entero, en tanto por ciento y, sin redondeo, con la parte decimal truncada) que le correspondería al valor introducido como parámetro, pero sin comprobar que dicho valor estaba dentro de la escala (de 0 a 1023). Para restringir los valores a un cierto rango, se podría utilizar la función "`constrain(valor, min, max);`".

[73] Para manejar la comunicación serie con I^2C se puede utilizar la librería Wire. Enlace: librería Wire.

[74] Arduino UNO gestiona las fuentes de referencia voltaica mediante la función "`analogReference();`". Se puede configurar una referencia interna de 1'1[V] (mediante `analogReference(INTERNAL);`), una referencia de 5[V] (con `analogReference(DEFAULT);`) o una referencia externa que se toma del valor de entrada en el pin AREF, de entre 0 y 5[V] (a través de `analogReference(EXTERNAL);`). Enlace: analogReference().

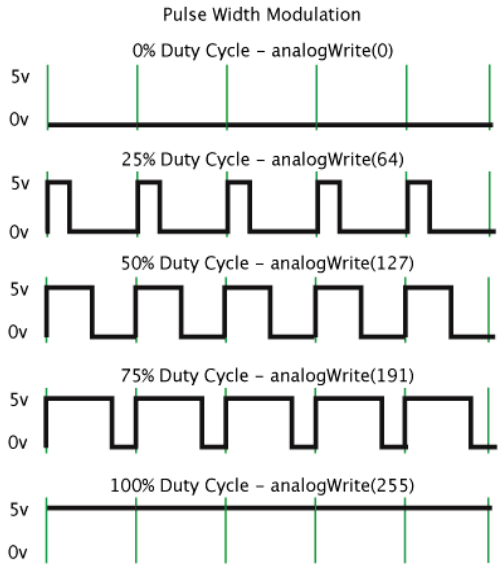
[75] Si se desea consultar el tutorial sobre las salidas analógicas (con PWM) que se presenta en el sitio web de Arduino, se puede acceder a él a través del siguiente enlace: http://arduino.cc/en/Tutorial/PWM.

[76] Las salidas analógicas de pueden manejar con la función `analogWrite()`. Enlace: analogWrite().

PWM (*Pulse Width Modulation*) es una técnica de **modulación basada en la variación de la anchura** (o amplitud) **del pulso de una señal** digital, calculada en base a una señal analógica dada. Dicha señal digital (de forma cuadrada) estará **formada por pulsos de frecuencia constante** (490 ó 500[Hz] aproximadamente) y, aunque es posible cambiar la frecuencia por defecto de la señal digital, la mayor parte de las veces no será necesario. Con la PWM, **a medida que la señal analógica varía su amplitud, la anchura del pulso de la señal digital cambia**. Al modificar la duración de los pulsos, se **modifica la tensión media** resultante. Cuanto más cortos (menos anchos) sean los pulsos (y así **más separados estén entre sí** en el tiempo), como su frecuencia es constante, **menor** será la **tensión media de salida**; y viceversa. Si duración del pulso varía mientras la señal se está emitiendo, la tensión promedio puede ir variando a lo largo del tiempo de forma continua. Cuando la duración del pulso coincida con el período de la señal (no habrá distancia entre pulso y pulso y por lo tanto sería una señal de valor constante), la tensión media de salida será la máxima posible, 5[V].

En Arduino UNO, los **pines** de salida PWM son: el pin **3**, el **5**, el **6**, el **9**, el **10** y el **11**; aunque todas las salidas digitales se podrían utilizar como salida PWM, en caso de necesitar más de las 6 mencionadas. Cada pin PWM tiene una **resolución de 8 bits**, es decir, que se pueden obtener 2^8 = **256 valores diferentes posibles** para indicar la duración deseada, de los pulsos de la señal digital generada a partir del valor analógico.

Si se establece el valor mínimo (0), se emitirán pulsos lo más estrechos posibles (de menor duración posibles) y se generará una señal analógica equivalente a 0[V]; mientras que si se establece el valor máximo (255), se emitirán pulsos de máxima duración y se generará una señal analógica equivalente a 5[V]. Cualquier valor intermedio (entre 0 y 255) emitirá pulsos de duración intermedia y generará una señal analógica de un valor entre 0 y 5[V]. La diferencia de voltaje que puede haber entre dos valores promedio contiguos (entre 0 y 1 x ejemplo), se puede calcular dividiendo el posible rango de voltajes de salida, entre el número de valores promedio. En Arduino UNO, como el rango de voltajes posibles va de 0 a 5[V] tenemos que: (5[V] – 0[V] / 256) = 19'5[mV]; es decir, cada valor promedio tiene una diferencia de voltaje con el anterior y con el siguiente de 19'5[mV].



HW Arduino UNO, imagen 14

La placa Arduino UNO incorpora 3 temporizadores diferentes (que pueden utilizarse como contadores) para controlar los pines PWM, de manera que se mantienen la frecuencia constante de los pulsos emitidos. El *timer1* (de 16 bits) maneja los pines 3 y 11; el *timer2* (de 8 bits), los pines 5 y 6; y el *timer3* (de 8 bits), los pines 9 y 10. Estos *timers* **tienen** un sistema de preescalado para su propia **señal de reloj**, que puede venir desde el sistema de reloj del microcontrolador, desde un reloj interno o externo, o desde pines externos. Los *timers* funcionan en paralelo a la CPU y de forma independiente a ella. Su **funcionamiento** básico consiste en **incrementar el valor** del registro **del contador, al ritmo que marca su señal de reloj**. Permiten configurar, tanto la frecuencia, como el ciclo o modo de trabajo.

ICSP (*In-Circuit Serial Programming*)

Se trata de un método de programación directa, de microcontroladores AVR, a través de un **programador *hardware* ISP** (*In-System Programming*)[77] externo que se conecta al cabezal ICSP de la placa para comunicarse con el microcontrolador que se vaya a programar.

HW Arduino UNO, imagen 15

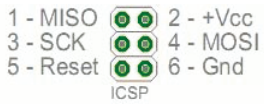
El programador *hardware* ISP es otro sistema integrado, capaz de escribir en la memoria Flash del microcontrolador, **a través del protocolo** estándar de datos en serie síncrono, **SPI** (*Serial Peripheral Interface*) ya explicado en apartados anteriores. Arduino UNO, en principio, no lo necesita, puesto que ya tiene el *bootloader* instalado en la Flash del ATmega328P; el *bootloader* sustituye al ISP conectado al ICSP. Pero se puede adquirir un programador *hardware* ISP en caso de que se quiera instalar, modificar, o eliminar el *bootloader*. Existen varios de modelos de ISPs de apariencia similar, aunque no todos son compatibles con los microcontroladores de AVR[78]. La versión más extendida es el que consta de conector **USB** (para enchufar al computador), clavija **ICSP** (para conectar al cabezal ISCP de la placa Arduino) y **microcontrolador** interno, especializado en su función de programador. Además del conector ICSP del ATmega328P, el micro ATmega16U2, que gestiona la conexión USB, también tiene su propio conector ICSP.

[77] Para programar el micro de Arduino a través del cabezal ICSP se necesita un programador *hardware* ISP específico que se ha de adquirir aparte. Pero además de conectar correctamente el programador ISP, se han de seguir una serie de pasos que hacen necesario un que determinado *software* se ejecute en el computador.
En este enlace se explica cómo utilizar el ISP: http://arduino.cc/en/pmwiki.php?n=Hacking/Programmer.

[78] Los programadores *hardware* ISP externos compatibles con el ATmega328P se pueden consultar en el fichero "programmers.txt" que se descarga junto con el IDE de Arduino. Son: "AVRISP mkII", "USBtinyISP", "USBasp", "Arduino as ISP", y "USB AVR Programmer".

El ICSP sirve, por tanto, para **grabar en el microcontrolador el *bootloader*** (porque se ha dañado o no lo tiene; de esta forma se han grabado los *bootloaders* en los microcontroladores de las placas Arduino que vienen de fábrica) simplificando así la escritura en memoria, puesto que el *bootloader* recibe el código máquina del programa en el IDE desde el puerto serie/USB y lo carga y escribe (graba) en la memoria. Pero sirve, además, para, si se desea, **grabar en el microcontrolador directamente los programas sin *bootloader***, lo cual deja más espacio en la memoria Flash del micro y permite ejecutar los programas inmediatamente después alimentar eléctricamente la placa, sin esperar a la ejecución del *bootloader*.

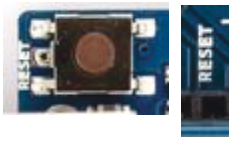
Existen dos estándares para los conectores ICSP, uno de 6 pines y otro de 10 (aunque el de 10 pines ya se encuentra obsoleto). **Arduino UNO incorpora**, para el ATmega328P, **un conector ISCP de 6 pines,** cada uno de los cuales está internamente conectado a una patilla del microcontrolador.



HW Arduino UNO, imagen 16

El **pin VCC (5V)** es para alimentación El **pin GND** es para tierra o masa. El **pin SCK** es el reloj que determina el ritmo al que se transfieren los datos (sus impulsos sincronizan la transmisión de datos generada por el maestro). El **pin RESET** activa o desactiva la comunicación y se encuentra conectado al pin SS del micro; así, cuando reciba un voltaje HIGH**,** no ocurrirá nada; y cuando reciba uno LOW, detendrá la ejecución del programa que haya en ese momento grabado, para recibir un nuevo programa. Para la comunicación serie son el **pin MISO**, para la salida serie de datos (enviar datos al maestro), y el **pin MOSI**, para la entrada serie de datos (enviar datos a los periféricos).

Reinicio de la placa



HW Arduino UNO, imagen 17

Existen varias opciones para reiniciar (detener y volver a arrancar) la placa Arduino UNO; todas ellas, consisten en reiniciar su microcontrolador. Ya sea por la desconexión y conexión de un cable **USB** entre el computador y la placa (internamente a través de un condensador), mediante el botón de *Reset* de la **placa**, o mediante el **pin** RESET. Además, Arduino permite desde su **IDE**, un *reset* automático.

Una de las formas más comunes de reiniciar Arduino UNO es mediante el **botón de *Reset*** de la placa, un botón que, al ser presionado, envía una señal LOW (0 voltios) al pin RESET de la placa, (pin que puede utilizarse al igual que el botón de *reset* y que es de utilidad, cuando, por ejemplo, se conectan sobre la placa Arduino *shields* o placas supletorias, que pueden ocultar e imposibilitar el uso del pulsador *reset* de la placa).

Como ya se ha explicado, en el momento de arranque de la tarjeta Arduino, se activa la ejecución del *bootloader* del microcontrolador. El botón *reset* se suele utilizar para permitir la carga (grabación) de un nuevo programa en la memoria Flash del microcontrolador (eliminando el que estuviera grabado anteriormente) y su posterior puesta en marcha. En la placa Arduino UNO no es necesario prácticamente nunca pulsarlo antes de cada carga, puesto que la placa está diseñada de tal forma que el *bootloader* se activa cómoda y directamente desde el IDE instalado el computador, presionando con el ratón en el icono correspondiente. Si no se produce el reinicio por alguna de estas causas anteriores, una resistencia preparada para ello, fija un valor de 5[V] en la línea y permite el funcionamiento normal del microcontrolador.

Arduino UNO también tiene un ***reset* automático**. Está diseñado de tal forma que pueda ser reiniciado por el *software* que se ejecute en el computador al que está conectado, con sólo pulsar el botón de correspondiente en el IDE de Arduino. Así, el *bootloader* puede tiene un breve tiempo de espera para no tener que requerir el pulsado del botón de *reset* de la placa antes de un proceso de carga del programa a ejecutar. El *reset* automático, permite además que, cada vez que se realiza una conexión de la placa Arduino UNO, a través de USB, a un equipo con Mac OS X o Linux, la placa se reinicie. Por ello, que hay que tener en cuenta que, si el programa grabado en el microcontrolador está pensado para, nada más iniciarse, recibir datos provenientes desde el computador a través de USB, nos tendremos que asegurar de que, el software encargado de enviarlos, espere al menos un segundo (porque el *bootloader*, al reinicio de la placa, también tarda un tiempo en ponerse en marcha, medio segundo aproximadamente) después de abrir la conexión USB para proceder al envío.

Para **deshabilitar su reinicio automático**, Arduino UNO contiene una traza (etiquetada como "RESET-EN" en el dorso de la placa) la cual se puede cortar (pudiendo volver a unir por soldadura las almohadillas de ambos lados de la traza para habilitar el *reset* automático de nuevo). También se puede desactivar mediante la conexión de una resistencia de 110[Ω] desde 5V hasta la línea de *reset*[79]. O conectando un condensador de 10 microfaradios entre los pines "GND" y "RESET" de la placa. En todo caso, una vez deshabilitado el auto-*reset*, el procedimiento de carga de los programas será: primero, mantener apretado el pulsador de reinicio de la placa Arduino; después, clicar en el icono "Upload" del IDE Arduino y a continuación, soltar el pulsador cuando se ilumine el led RX de la placa; finalmente, la carga debería empezar haciendo parpadear los leds RX y TX.

[79] En el foro de Arduino se pueden consultar los detalles (ver este hilo del foro).

Software

Arduino UNO, al igual que el resto de placas de Arduino, **se puede comenzar a programar** a través de su IDE[80]. Basta con **descargar el IDE** (en un computador con un sistema operativo Windows, Mac OS X, o Linux, al cual esté conectada la placa Arduino UNO mediante un cable con conector USB), ir a "Herramientas\Tarjeta" y seleccionar la placa Arduino UNO, e ir a "Herramientas\Puerto Serial" y seleccionar el puerto de comunicación ("COM"), habilitado para la tarjeta Arduino (en Windows generalmente es el COM4).

Cuando se programa la placa, se está programando su microcontrolador. El ATmega328P de Arduino UNO, tal y como se ha explicado anteriormente, ya viene con un *bootloader*[81] instalado que permite cargar desde el IDE nuevo código en la placa **sin necesidad de un programador de hardware externo** (ISP) conectado al ICSP (del ATmega328P) de la placa; aunque en caso de que se desease, se podría pasar por alto el *bootloader* instalado y programar el microcontrolador directamente a través de su cabezal ICSP[82].

Para entender el lenguaje de programación de Arduino, se puede consultar la Referencia del Lenguaje[83] en el sitio web de Arduino. Además, al final de este documento, se presenta un anexo extraído de la web de Arduino, que también puede consultarse en caso de que se desee, a modo de resumen de los principales elementos que conforman el lenguaje de programación de Arduino.

[80] El IDE se puede descargar del sitio web de Arduino (Enlace: descarga). Para más detalles, se puede consultar la Referencia del Lenguaje de Arduino (ver referencia) y los tutoriales (ver tutoriales).
[81] Los detalles del *bootloader* se encuentran el tutorial del sitio web de Arduino (ver gestor de arranque).
[82] Para programar Arduino sin *bootloader*, sino directamente mediante el conector ICSP y un hardware externo ISP, como ya se explicó en el apartado "Comunicación serie" de este documento, se puede consultar del sitio web de Arduino para programación mediante ISP y el ICSP (ver instrucciones).
[83] La Referencia del Lenguaje de Arduino se encuentra en: http://arduino.cc/en/Reference/HomePage.

SENSOR ULTRASÓNICO

Sensores

Un **sensor** (del latín *sentĭo*, sentir) es un **dispositivo** electrónico diseñado para recibir información. **Detecta una** determinada **acción** externa **y la transmite** adecuadamente. Se encarga, para ello, de **transformar** las **magnitudes** físicas o químicas (variables de instrumentación como la temperatura, la presión, la distancia, el desplazamiento, la aceleración, inclinación, etc.), **en magnitudes**, normalmente **eléctricas**, que seamos capaces de cuantificar y manipular.

De manera cada vez más frecuente, podemos encontrar sensores aplicados a cualquier área tecnológica y en muchos de los elementos que nos rodean (semáforos, teléfonos móviles, industria, vigilancia, etc.). Los dispositivos que incorporan sensores reaccionan a la información que reciben de ellos. De esta forma, los sensores **permiten interactuar con el entorno**, aportando información de ciertas variables que nos rodean, para procesarlas, generar órdenes o activar procesos.

Suele ser habitual que la **señal de salida** de un sensor, no sea la adecuada para ser procesada por los circuitos actuadores, y sea necesario **adaptarla y amplificarla**. Además, como también suele ocurrir, si depende de condiciones como la temperatura o la tensión de alimentación, también es preciso **linealizar** el sensor y **compensar** sus variaciones mediante **acondicionadores** de señal.

Los microcontroladores son un tipo de acondicionadores de señal económicos, versátiles, y capaces de procesar prácticamente cualquier tipo de señal. Constan de unas **entradas y salidas (E/S) digitales que pueden recibir señales y entregar señales** a otros acondicionadores de tipo de puente de resistencias, transistores y operacionales. Además **pueden implementar convertidores de señal ADCs programables**, entradas de captura y cuenta de pulsos, y líneas de comunicación serie I^2C. Los más extendidos comercialmente, son los del fabricante Microchip y los de Atmel[84].

A la hora de elegir un sensor[85], para utilizarlo en una aplicación concreta, se han de tener en cuenta aspectos que permitan conseguir el mejor **rendimiento** para dicha aplicación, tales como la

[84] Atmel es el fabricante del ATMega328P que incorpora la placa Arduino UNO.
[85] Existe en el mercado gran variedad de sensores. Para mayor información sobre los distintos tipos de sensores y sus clasificaciones más comunes, se puede consultar el Anexo II, al final de este documento.

rapidez de respuesta, la situación en la que utilizarán, el radio de acción que han de tener, la fiabilidad que se espera para su correcto funcionamiento, las tensiones de alimentación, el consumo de corriente, los márgenes de temperatura de su funcionamiento, las posibles interferencias producidas por agentes externos, su resistencia a dichos agentes externos o la relación calidad precio. Además, es necesario conocer algunas propiedades del sensor, que dan información en cuanto a la **eficacia** que puede tener, principalmente la resolución, la sensibilidad, el grado de error, de precisión o de repetitividad, entre otros.

De manera habitual, **en función de la aplicación** que se vaya a dar al sensor, normalmente se elige **un sensor de un tipo u otro**. Por nombrar los más comunes en este sentido, se puede decir que en iluminación, por ejemplo, se suelen utilizar sensores fotorresistivos o sensores fotoeléctricos; sin embargo, para trabajar con la temperatura, se utilizan los termistores. En medición de humedad, se pueden aplicar los sensores resistivos o los sensores capacitivos. Para medir posiciones o inclinación, es habitual emplear sensores mecánicos, sensores resistivos, acelerómetros o sensores magnéticos. Relacionados con la presión, están los sensores piezoeléctricos o los sensores resistivos. Para medir caudal, existen sensores piezoeléctricos o magnetorresistivos. En detección de presencia, una buena elección son los sensores magnéticos, los sensores de infrarrojos o los sensores de ultrasonidos. Y **para medición de distancias** (objetivo de este proyecto) se puede optar por servirse de sensores de infrarrojos (*Infrared Radiation,* IR) o **de sensores de ultrasonidos** (*ultrasonic rangers*).

Entre los **sensores de infrarrojos** para medición de distancias compatibles con Arduino, se encuentran los IR analógicos de la gama **GP2** de Sharp. Los sensores IR GP2, están formados por un emisor y un receptor de infrarrojos. Cuando el haz de infrarrojos enviado por emisor, impacta sobre un obstáculo, rebota en el obstáculo e incide sobre el receptor con un determinado ángulo a partir del cual se podrá calcular la distancia a dicho obstáculo, según el voltaje que devuelve el sensor (a través de su conector analógico), el cual se muestra en una gráfica de la hoja de especificaciones del fabricante del sensor, cuyo valor será necesario calibrar, en cada caso particular. Otros sensores de infrarrojos como los QRD (1113/1114)[86], similares a los GP2 en cuanto a modo de funcionamiento, han sido diseñados, más que para medición de distancias, para detección de presencia en distancias cortas, puesto que su rango de detección va solamente de 0 a 3[cm]. Por otra parte, para medición de distancias a través de Arduino, los sensores por excelencia (en cuanto a rendimiento, eficacia y simplicidad a la hora de programarlos con el IDE Arduino) son los de ultrasonidos. Se explican con más detalle a continuación.

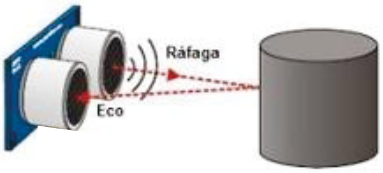
[86] La hoja de especificaciones del QRD (113/114) del fabricante QT Optoelectronics, se puede encontrar en http://pdf.datasheetcatalog.com/datasheet/QT/QRD1113.pdf. La hoja de especificaciones del QRD (113/114) de Fairchild Semiconductor, está disponible en https://www.fairchildsemi.com/datasheets/QR/QRD1114.pdf.

Sensores de ultrasonidos

Los ultrasonidos son señales acústicas (vibraciones mecánicas en forma de ondas elásticas longitudinales) que se propagan en un medio elástico (como el aire) con frecuencias que **superan el rango de las frecuencias perceptibles por el oído humano**, aproximadamente a partir **de los 20[kHz]** (hasta más de 1[GHz]). Debido a su naturaleza, pueden sufrir fenómenos de reflexión, refracción y difusión. Suelen emplearse en **medición de distancias** o **detección de presencia.** Y requieren un dispositivo **emisor,** un dispositivo **receptor** y un dispositivo de **medición.**

Cuando se emiten ultrasonidos, los cuales se propagan a la velocidad constante del sonido (340[m/s] a 25[°C] de temperatura, ó 343[m/s] a 20[°C]), **y chocan** perpendicularmente con un obstáculo, **rebotan, a la misma velocidad, pero en dirección contraria** (normalmente con un ángulo, por lo que se ha de introducir un factor de corrección). Los sensores ultrasónicos se sirven de esta propiedad y calculan el tiempo que tardan las ondas ultrasónicas desde que son emitidas por el propio sensor hasta que se recibe su rebote.

Para generar los ultrasonidos, los sensores ultrasónicos utilizan **campos eléctricos variables para,** basándose en un efecto piezoeléctrico de deformación que experimentan ciertos cristales al introducirse en un campo eléctrico, **hacer oscilar un cristal** (normalmente cuarzo, y a una frecuencia de 40[KHz]). Generalmente están **formados por dos transductores** en forma de cápsulas cilíndricas (uno **emisor** de ultrasonidos y, otro, **receptor** del rebote o eco de dichos ultrasonidos) **y conectores** o pines (de tipo macho) **de entrada o salida** digital.



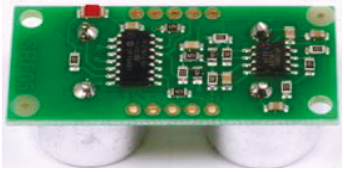
Sensor ultrasónico, imagen 1

Los sensores de ultrasonidos han sido **diseñados para calcular el tiempo transcurrido entre la emisión y** posterior **recepción de los ultrasonidos que envía**. Así, una vez obtenido dicho tiempo y conociendo la velocidad de propagación de los ultrasonidos en el aire, puede calcularse (estimarse) la distancia entre el sensor y el objeto que se encuentra delante de él y que ha provocado el rebote de los ultrasonidos que envió.

El funcionamiento de un sensor de ultrasonidos, se resume a continuación. En primer lugar, el **emisor** (cápsula emisora) del sensor genera y envía una **señal acústica ultrasónica**, normalmente de 40 [kHz]. La emisión debe ser corta (entre unos 10 y 600[µs]), para evitar grandes trenes de señal que puedan interferirse entre sí y distorsionar los ecos producidos en los rebotes de la señal ultrasónica. Después, transcurre un **tiempo de espera** de seguridad (de 2[ms] aproximadamente,

dependiendo del número de cápsulas cilíndricas del sensor), que evite lecturas falsas en distancias cortas. Sucedidos los tiempos de emisión y seguridad, se pondrá en actividad un circuito asociado al **receptor** (cápsula receptora), que **detecta** internamente **si** en algún momento **se recibe una señal acústica ultrasónica**. Según la distancia máxima de medición para la que esté diseñado el sensor de ultrasonidos (sus cápsulas transdustoras), se debe **esperar un tiempo** determinado, a partir del cual, **en** caso de que el receptor no reciba el eco de la señal ultrasónica que envió el emisor, el emisor interpretará que no había un obstáculo en el camino **y** procederá de nuevo a **emitir otra señal ultrasónica**. Este tiempo de espera también puede depender de otros factores tales como la temperatura; para una temperatura de 25[°C], suele estar en torno a los 23[ms], para un alcance de 4[m] (8[m] de ida y vuelta). A través de la programación de un microcontrolador y conociendo la fórmula de la distancia (d = ½*v*t) e interpretando las señales de voltaje que proporciona el sensor, ya se puede **calcular la distancia** hasta un objeto u obstáculo, una vez recibido el eco del ultrasonido que envió el emisor.

En el mercado, se pueden encontrar diversos sensores de ultrasonidos para medición de distancias. Los de mayor rango de detección en distancias cortas y más comúnmente utilizados, **compatibles con Arduino,** son el SRF05, el sensor PING))) y el sensor HC-SR04. Tienen en común que los tres son sensores de ultrasonidos **digitales**. Existen también sensores ultrasónicos analógicos (como el LV-EZ0), pero no permiten medición de distancias inferiores a 30[cm].


Sensor ultrasónico, imagen 2

El **sensor SRF05** de Devantech[87] es un sensor digital capaz de medir distancias de entre 0'03 y 3[m]. Está formado por cinco agujeros conectores en su parte inferior, que permiten unirlo a la placa Arduino. Es **posible conectarlo al Arduino** a través de cuatro cables (como en su versión anterior, SRF04) o con tres. Si se utilizan **cuatro cables** para conectarlo a la placa Arduino, los conectores del sensor, de izquierda a derecha funcionarían como: **5V** (alimentación), **Echo** (para conectarlo a un pin de entrada digital de Arduino, que reciba la señal a nivel alto indicando la recepción de ultrasonidos por parte del sensor), **Trig** (para conectarlo a un pin de salida digital de Arduino, enviar un pulso de 10 microsegundos al sensor y que entre en funcionamiento, emitiendo ultrasonidos), un **conector al que no se ha de conectar** nada, **y GND** (tierra). En caso de utilizar sólo **tres cables**, los conectores del sensor, de izquierda a derecha funcionarían como: **5V** (alimentación), un **conector al que no se ha de conectar nada**, un **conector que funcionará alternándose como Trig** (pin digital de salida en el Arduino) **y como Echo** (el mismo pin digital, esta vez, entrada en el Arduino) y dos conectores para GND (tierra).

[87] El fabricante Devantech, ofrece el sensor SRF05 (y otras variantes del mismo, como el SRF08 o el SRF10) en su sitio web: http://www.robot-electronics.co.uk/acatalog/Ultrasonic_Rangers.html.

Sensor ultrasónico, imagen 3

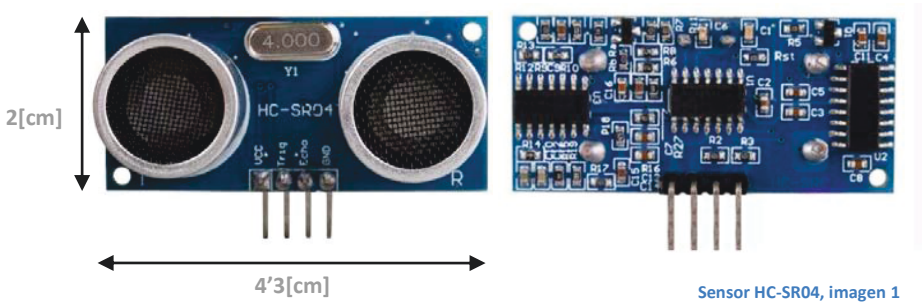
El **sensor PING)))** de Parralax[88], es otro sensor digital capaz de medir distancias de entre 0'03 y 3[m]. Consta de tres patillas conectoras: GND (tierra), 5V (alimentación) y SIG (señal). La patilla **SIG** del sensor será **tanto la entrada** digital (para activar el sensor desde un pin digital de salida de la placa Arduino, a través del cual se ha de enviar al sensor un pulso a nivel alto de 5 microsegundos) **como la salida** digital (para emitir un pulso, que se enviará a al mismo pin digital de la placa Arduino, que ahora será de entrada, indicando que el sensor ha recibido el rebote del ultrasonido, para que a través de Arduino se puedan empezar a medir las distancias). Por lo tanto, el pin digital de la placa Arduino, al cual esté conectada la patilla SIG del sensor PING))), habrá de comportarse (programarse) primero como salida (OUTPUT) y después como entrada (INPUT).

El sensor **HC-SR04**, similar a los anteriores y de precio más económico (unos 5€, respecto a los aproximadamente 30€ del PING))) o los SRFxx), ha sido finalmente el que se ha elegido para utilizarlo en este proyecto. Se explica en las próximas líneas.

Sensor HC-SR04

Para obtener información acerca del mismo, se ha de consultar la hoja de especificaciones[89] (*datasheet*) que proporcione su fabricante o distribuidor acerca de las características y el modo de funcionamiento de este sensor.

El sensor HC-SR04 es de color azul y plateado (como la tarjeta Arduino). Sus dimensiones son de 4'3 x 2 x 1'5[cm] aproximadamente. Y su masa de unos 10[g]. Tiene este aspecto:

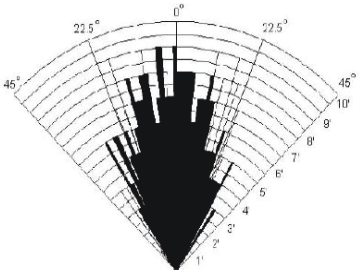


Sensor HC-SR04, imagen 1

[88] El sensor PING))) de Parralax se puede encontrar en: http://www.parallax.com/product/28015.
[89] El sensor HC-SR04 se puede conseguir a través de diversos distribuidores, como pueden ser:
Opiron (opiron.com/tienda/es/sensores/11-sensor-ultrasonidos-hc-sr04.html),
ElecFreaks (elecfreaks.com/store/hcsr04-ultrasonic-sensor-distance-measuring-module-ultra01-p-91.html),
AccuDIY (http://accudiy.com/index.php?main_page=product_info&cPath=22&products_id=184).

Está formado esencialmente por transmisores y receptores ultrasónicos y circuitería interna de control. Las características principales del sensor HC-SR04 se pueden resumir así:

- **Voltaje** (o tensión) de funcionamiento: 5[V] DC (mínimo 4'5 y máximo 5'5).
- **Corriente** de trabajo (o **consumo en funcionamiento**): en torno a 15[mA] (mínimo 1[mA] y máximo 20[mA]).
- Corriente estática (o consumo en reposo): normalmente < 2[mA] (como mínimo 1'5[mA], y como máximo 2'5[mA]).
- **Frecuencia** de las ondas ultrasónicas que emite: 40[kHz].
- **Ángulo** de medición: 30°, efectivo inferior a **15°**.
- **Detección de distancia: de 2[cm] a 4'5[m].**
  A más de cuatro metros puede haber detección, pero no se garantiza una buena medición.
- **Precisión**: puede variar **entre los 2 ó 3[mm]**.
- **Señal de salida** (*trig*): pulso inicial de 10[µs] a nivel TTL.
- **Señal de entrada** *(echo)*: pulso a nivel TTL de una duración igual al tiempo de ida y vuelta de los ultrasonidos desde el sensor, hasta el obstáculo al cual se desea medir la distancia.



Practical test of performance, Best in 30 degree angle

Sensor HC-SR04, imagen 2

Los pines de conexión del HC-SR04 a través de los cuales se ha de conectar a la placa Arduino (a través o no de la *protoboard*, que no tiene por qué ser necesaria) son: el **pin VCC** (+5[V] DC) se unirá a la salida de 5V de la tarjeta Arduino que suministrará alimentación eléctrica al sensor para su funcionamiento; el **pin Trig,** para enviar la señal de t*rigger* (de emisión), se conectará al pin digital de la placa Arduino encargado de la emisión o disparo del ultrasonido; el **pin Echo,** para la señal de *echo* (de recepción), irá unido al pin digital de entrada de la placa Arduino destinado para la recepción del ultrasonido; y el pin **GND**, a la toma de tierra de la tarjeta Arduino.

El sensor HC-SR04 funciona como lo haría un sónar. Permite estimar la distancia a un punto, a través del sistema de medición de tiempo que incorpora, mediante el cual, **calcula la diferencia de tiempo** que se sucede **entre que transmite y recibe los pulsos** digitales **que envía y captura** a través de los transductores cilíndricos que lleva acoplados a su superficie. En los cronogramas de las hojas de especificaciones se gráficamente y se explica cómo han de ser dichos pulsos.

DISEÑO DEL PROGRAMA

Definición del Sistema

En el desarrollo de la aplicación de medición de distancias objetivo de este proyecto, se ha empleado una placa Arduino de la gama UNO (la más extendida) a la cual se ha conectado el sensor ultrasónico HC-SR04 (el más comúnmente empleado en medición de distancias, actualmente). Para que haya comunicación entre dos dispositivos, la entrada de uno (la placa Arduino UNO) tiene que ir conectada a la salida del otro (el HC-SR04), y viceversa. Además, en una comunicación asíncrona, tendrán que comunicarse a la misma velocidad.

Arduino UNO, cuando se conecta mediante un cable con conector USB a un computador (Windows, Mac OS X, o Linux) con el IDE Arduino, se comporta ante el computador, como un puerto serie a través del cual la placa y el computador pueden transmitirse información entre sí. El sistema de transmisión de información en este caso es una comunicación serie UART (*Universal Asyncronus Receiver/Transmiter*) a través de la cual, la comunicación serie se realiza de forma asíncrona, es decir, sin una señal de reloj que indique determine la velocidad de transmisión de información (cada cuanto tiempo se realizan las lecturas o escrituras) y sincronice la transmisión de información entre la placa Arduino UNO y el computador. La placa y el IDE Arduino instalado en el ordenador, tendrán que coincidir en la velocidad a la cual se lleva a cabo la transmisión de información, determinada por los **baudios** o [bps] (unidades de señal). En esencia, **cuanto mayor es el número de baudios** elegido, **menos tiempo se sucede en la transmisión de información** y, por lo tanto, **más información se puede transmitir** respecto al tiempo. **En términos de bits,** si cada baudio equivale a 1 bit, los baudios indicarán los **bits que se transmiten por segundo**. De esta forma, se puede calcular cuánto tiempo lleva la transmisión de 1 bit. Así, si se configura una comunicación con una velocidad de transmisión de 9600 baudios (la establecida por defecto en el monitor serial del IDE, y la más habitual), 1 bit será transmitido aproximadamente cada 104'17[µseg] (ya que 1[seg] / 9600[baudios] = 1'0417*10^{-4}[seg]).

En este proyecto, Arduino tendrá dos tareas principales: **medir la distancia** desde el sensor hasta un obstáculo (gracias al sensor ultrasónico HC-SR04) y **comunicar** al ordenador **dicha distancia** (mostrándola en el monitor serie del IDE de Arduino). Pero como no realiza estas tareas de forma simultánea, cuanto más tiempo se ocupe en una, menos tiempo tendrá para realizar la otra, es decir,

cuántos más sean los bits que se transmiten por segundo (**a más baudios en el *sketch* y el monitor *serial* del IDE**) entra la placa y el IDE, **mientras la placa y el sensor estén tomando** las medidas de **la distancia, menos tiempo** tardarán **en transmitir dicha información** al ordenador (y mostrarlo en el monitor *serial*). Para este proyecto, se ha decido utilizar el máximo de baudios posible en Arduino, que son 115200[bps], una velocidad de transmisión que permite visualizar de forma pausada cada una de las medidas de la distancia tomadas, en el monitor serial del IDE.

Alcance

- **Detección de distancia**: de 2[cm] a 4'5[m]. A más de 4[m] puede haber detección, pero no se garantiza una buena medición.
- **Precisión**: puede variar, **entre los 2 ó 3 [mm]**.
- **Ángulo** de medición: 30° (efectivo: < **15°**)

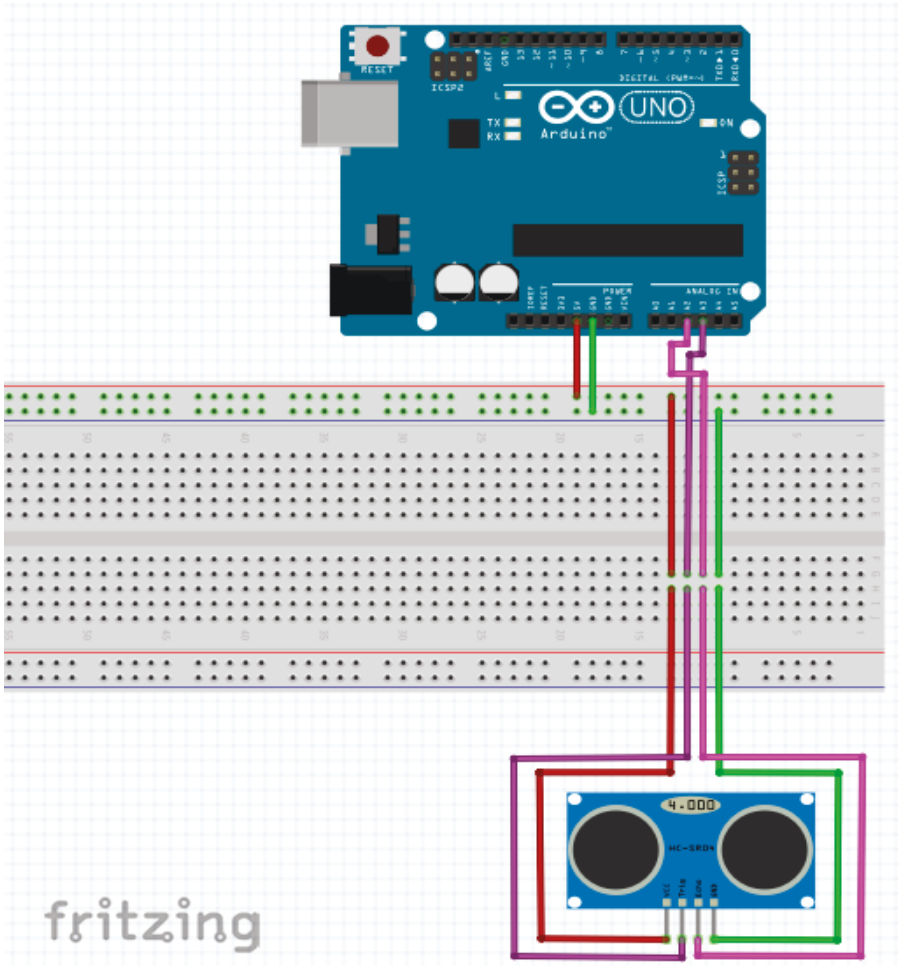
Restricciones

- **Voltaje** de funcionamiento: **mínimo 4'5[V], máximo 5'5[V]**. Normalmente 5[V]
- **Corriente**: estática o de reposo: **mínimo 1'5[mA], máximo 2'5[mA]**.Normalmente < 2[mA].
- **Corriente de trabajo**: **mínimo 1[mA], máximo 20[mA]**. Normalmente 15[mA].
- **Frecuencia** de trabajo (frecuencia de las ondas ultrasónicas): **40[kHz].**
- Señales de entrada y salida (*trigger* y *echo*) **a nivel TTL.**
- **La señal salida** o disparo, ha de una señal (**pulso**) de alto nivel (**HIGH**) **de**, al menos, **10[µs].**

Esquema del circuito

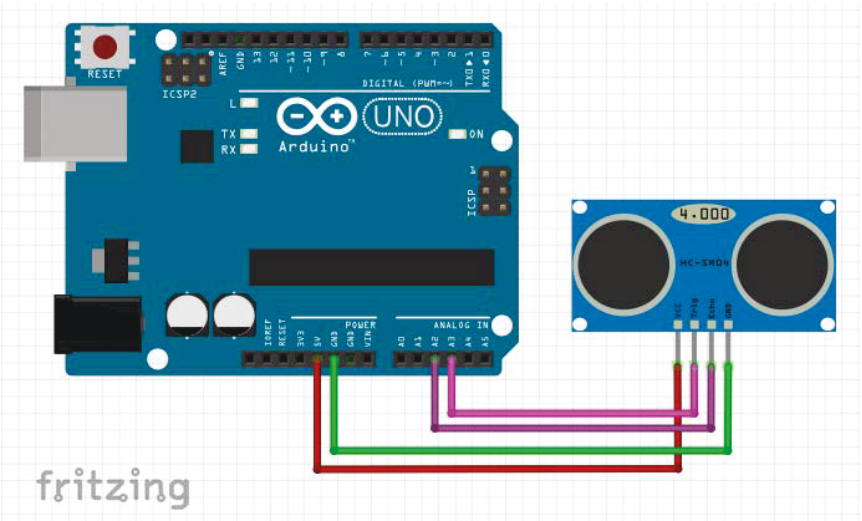
Para mostrar de manera gráfica el diseño del circuito se ha empleado Fritzing, un software de apoyo para el diseño de circuitos con Arduino. Permite representar gráficamente las conexiones en la *protoboard* y así, los montajes son fácilmente entendibles. Además, ofrece todo lo necesario para transformar el proyecto en una PCB real, para por ejemplo, crear nuevos *sields*.

Siempre que las conexiones se hagan de forma correcta, existen infinidad de posibilidades de diseño del circuito para implementar la medición de distancias por ultrasonidos con Arduino UNO y el sensor HC-SR04. La que aquí se muestra es la que se ha utilizado en el **diseño del *sketch* que se ha desarrollado** para este proyecto. El pin analógico A3 de Arduino UNO, se ha configurado como un pin digital, conectado al pin Trig del HC-SR04. El pin analógico A2 de Arduino UNO, se ha configurado como un pin digital, conectado al pin Echo del HC-SR04. El pin 5V de Arduino UNO, se ha unido al pin Vcc del HC-SR04. El pin GND de Arduino UNO, se ha unido al pin GND del HC-SR04.



Diseño del programa, imagen 1

Los pines Trig y Echo del sensor HC-SR04 podrían haberse conectado a cualquier pin digital de Arduino UNO. Además en este circuito, no tendría por qué ser necesario utilizar la *protoboard* (se ha utilizado, porque permite conectar las cuatro patillas del sensor HC-SR04 a ella y de esta forma se mantiene siempre en la misma posición sobre ella, lo cual es permite hacer de manera más factible las posteriores pruebas de medición de distancias). El esquema que se ha mostrado en la imagen anterior, sin utilizar la *protoboard*, podría simplificarse de la siguiente forma.



Diseño del programa, imagen 2

IMPLEMENTACIÓN DEL PROGRAMA

Entorno de desarrollo

Instalación del IDE Arduino

Se ha de programar el Arduino para que reaccione cuando recibe un impulso eléctrico desde alguna de sus entradas, y ofrezca algo a través de sus salidas. Para ello, como ya se ha comentado anteriormente, existe un **Arduino proporciona un IDE oficial** que facilita el desarrollo del código y está disponible para Linux, Windows, Mac OS X. Se puede **descargar de forma gratuita**[90]**. La última versión estable es Arduino 1.0.5** (r2, adaptada para Windows 8) así que se ha optado por ella para realizar este trabajo.

Para el desarrollo de este proyecto, se ha utilizado un **PC con el sistema operativo Windows** (versión Windows 7 Professional) actualmente el más extendido. En este caso, al descargar el fichero ejecutable de Arduino ("Windows ZIP file") y descomprimirlo se obtienen los siguientes directorios: **drivers** (controladores, específicos del tipo de sistema operativo, que permiten la comunicación de la tarjeta Arduino con el IDE Arduino), **examples** (ejemplos de código, *sketches*, que se pueden consultar y probar), **hardware** (herramientas e información sobre el *hardware* de Arduino, que utilizará el propio IDE), **java** (entorno de ejecución Java, JRE, para utilizar el IDE), **lib** (librerías Java y gráficos de configuración de la apariencia del IDE), **libraries** (librerías de Arduino para simplificar la programación de ciertos componentes), **reference** (documentación sobre las referencias del lenguaje de programación de Arduino) y **tools** (herramientas para IDE de Arduino). Todos ellos se encuentran junto a la propia **aplicación Arduino** (con sus extensiones) y un documento de texto, **revisions** (con las revisiones del IDE Arduino).

Lo siguiente que se ha realizado, siguiendo las instrucciones correspondientes, es **conectar el Arduino al PC**. Inmediatamente, en la placa Arduino UNO se activará el led On (que se mantendrá

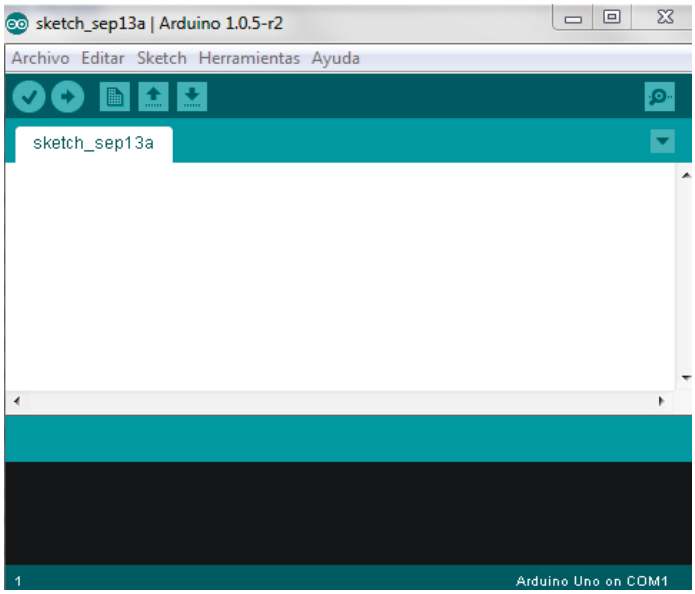
[90] El IDE Arduino se puede descargar desde su sitio web en http://arduino.cc/en/Main/Software.
Todos los pasos para realizar la descarga del IDE en general, en un computador con cualquier sistema operativo (Windows, Mac OS X o Linux) se encuentran en http://arduino.cc/en/Guide/HomePage; y en particular, para Windows, se encuentran en: http://arduino.cc/en/Guide/Windows.

encendido en color verde) y el led L (parpadeando en color naranja); en el PC aparecerá un mensaje de alerta indicando que el proceso de instalación no se ha realizado con éxito.

En el PC, hay que dirigirse a Panel de Control>Seguridad y Sistema>Sistema>Administrador de dispositivos. Aparecerán una serie de dispositivos controladores del PC y un dispositivo todavía desconocido para el PC (la placa Arduino UNO). Pinchamos con el botón derecho del ratón sobre ese dispositivo desconocido y seleccionamos "**Actualizar software de controlador**…" A continuación seleccionamos en la opción "Buscar software de controlador en el equipo" y "Examinar". Vamos directorio donde habíamos descargado el fichero del IDE que descomprimimos. Seleccionamos el directorio Drivers (incluyendo las subcarpetas), pasamos al siguiente paso y aceptamos la instalación. Se puede comprobar que en el PC, en "Administrador de dispositivos", aparece Arduino Uno (COM4), es decir que se han instalado los *drivers* adecuados que permitirán que el puerto USB conectado al PC y a la placa Arduino aparezca como un puerto serie o "COM" dentro del PC, para que se pueda llevar a cabo la comunicación con la placa Arduino, ahora conectada a ese puerto del PC. **Ya se puede empezar** a utilizar Arduino.

Ejecución del IDE Arduino

Para ejecutar el entorno de desarrollo, simplemente hay que dirigirse al directorio donde se descomprimió el IDE Arduino y seleccionar el fichero la aplicación Arduino (**Arduino.exe**). Se abrirá una **ventana** con un aspecto similar al que muestra la siguiente imagen, **donde podremos empezar** a escribir **el** código del programa ejecutable por Arduino o ***sketch***.



Implementación del sketch, imagen 1

Para comenzar con la programación de la placa a través del IDE Arduino, se debe **comprobar** en la ventana del IDE, en el menú Herramientas>Tarjeta, que **la placa seleccionada** es Arduino UNO, y en menú Herramientas>Puerto serial, **seleccionar el** mismo **puerto COM** que el **que se habilitó en la instalación de los drivers para la comunicación serie** entre la placa y el computador (COM4).

Apariencia del IDE Arduino

La ventana principal del IDE se encuentra dividida en cinco secciones que, de arriba hacia abajo son: una barra de **menús**, una sección de **botones**, un **editor** de texto, una sección y **consola** de mensajes y una barra de **estado**. Se explican con más detalle a continuación.

En la parte superior del IDE aparece la barra de menú que, para complementar las funciones que se pueden realizar con la sección de botones, se divide en cinco entradas principales:

- **Archivo** (*file*). Ofrece distintas acciones, algunas estándar (**Nuevo**, **Abrir**, **Guardar**, **Cargar** en la memoria del micro, **Imprimir** y **Cerrar**) y otras específicas, como:
    - ***Sketchbook***. Librería de los *sketches* guardados en sus correspondientes subcarpetas. Esta carpeta, se genera automáticamente la primera vez que se ejecuta el IDE y se encuentra ubicada dentro de la carpeta personal de usuario del sistema. Dentro de la misma, se crea una subcarpeta diferente para cada proyecto, donde se guardarán sus *sketches* correspondientes.
    - **Ejemplos** de *sketches* ordenados por tipo, que vienen con el IDE y se pueden probar y utilizar.
    - **Cargar usando un programador** ISP externo (para programar directamente el micro, sin *bootloader*) seleccionándolo de entre la lista de programadores ISP compatibles que aparecen en el menú de Herramientas/Programador.
    - **Preferencias**, para cambiar características del IDE como el idioma, tamaño de fuente, ubicación de la carpeta *Sketchbook*, nivel de detalle en los mensajes mostrados en la consola durante la compilación o carga de los *sketches*, etc. Otras preferencias que no aparecen en este menú, también se pueden modificar, "a mano" y mientras no esté ejecutándose el IDE, en el fichero de "preferences.txt" que se encuentra en una carpeta cuya ubicación se muestra en el propio cuadro emergente.
- **Editar** (*edit*). Permite realizar acciones típicas de cualquier editor de textos (como pueden ser **copiar**, **pegar**, **seleccionar**, **comentar**, **buscar** o buscar y **reemplazar**) y otras interesantes:

- **Copiar como HTML**, para pegarlo en páginas web.
- **Copiar para el Foro**, para pegarlo en el foro oficial de Arduino y poder recibir ayuda de la comunidad Arduino.

- **Sketch**. Ofrece distintas opciones relacionadas con el *sketch* con el que se está trabajando. Opciones para **verificar** (y compilar) el código, **abrir la carpeta** donde se encuentra el *sketch* "*.ino" que se está editando, **añadir una nueva pestaña** "sketch.ino" dentro del proyecto del *sketch* actual, o **importar librerías** (para reutilizar código de las librerías oficiales que ofrece Arduino o instalando las realizadas por nosotros o por un tercero) que en el sketch aparecen al principio del código del programa, como "`#include <librería.h>`".

- **Herramientas** (*tools*). En este menú se ofrecen distintas herramientas como
  - **Formato automático**, que organiza y ordena el código, con el sangrado adecuado, para que sea más legible.
  - **Archivar el sketch**. Crea un fichero comprimido con todos los códigos fuentes que corresponden al *sketch.* Es útil para portar *sketches* o realizar *backups*.
  - **Monitor serie.** Muestra el monitor serie igual que el botón que hay para ello.
  - **Tarjeta**. Muestra una serie de placas entre las cuales tenemos que seleccionar la que queremos utilizar para probar el *sketch.*
  - **Puerto serie**. Indica los puertos serie disponibles. Se ha de seleccionar el mismo al que se encuentre conectada la placa sobre la que se va a trabajar.
  - **Programador**. Por si se quiere programar directamente el microcontrolador. Se ha de seleccionar el programador que se va a utilizar, de entre la lista de los compatibles.
  - **Grabar secuencia de inicio** (*burn bootloader*). Permite grabar el *bootloader* de Arduino en el microcontrolador, por si se desea grabar uno nuevo *bootloader*.

- **Ayuda** (*help*). Permite acceder *off-line* (es decir, sin conexión a Internet) a varias secciones del sitio web de Arduino, con artículos, tutoriales y ejemplos que pueden servir de ayuda.

En la misma ventana del IDE, justo debajo de la barra de menú y antes del espacio en blanco para empezar a editar el programa, aparecen una serie de botones:

- **Verificar** (*verify*). Es el primer botón que se debe pulsar para probar cualquier modificación que se haga en el sketch, ya que comprueba que la sintaxis del código es correcta y si lo es, lo compila.

- **Cargar** (*upload*). Es el botón que se pulsa después de guardar y verificar el código (y una vez comprobado que el modelo de Arduino y el puerto al que está conectado son los correctos). Al pulsarlo, se activa el *bootloader* del microcontrolador (que se reinicia automáticamente) e, invocando internamente al comando "avrdude", se envía (**carga**) **el programa**, anteriormente verificado y compilado, **a la** memoria del micro de la **placa** Arduino que tenemos conectada al computador.

- **Nuevo** (*new*): abre un nuevo sketch en blanco.

- **Abrir** (*open*): muestra, en un menú, los sketches disponibles para abrir (los de ejemplo, o los propios creados anteriormente y disponibles en el *sketchbook*).

- **Guardar** (*save*). Guarda el programa en la carpeta de Arduino si no se le especifica otro directorio. El *sketch* (con extensión ".ino"), se guardará en una carpeta con el mismo nombre que el del *sketch*, que será la carpeta del proyecto, dentro de la cual, se podrán guardar posteriormente más *sketches* del mismo proyecto.

- **Monitor Serial** (*serial monitor*): sirve para depurar el programa. Se pueden introducir datos para enviárselos a Arduino, mostrar los mensajes enviados por el puerto serie o USB desde Arduino, activar o no el desplazamiento automático del área de salida de mensajes según éstos vayan saliendo, controlar el comportamiento de la nueva línea o retorno del carro (con las distintas opciones del selector), y configurar la velocidad de comunicación, en baudios (que son el número de cambios de estado de los bits por segundo. Es decir, que la opción de 9600 baudios, por ejemplo, significa que cada segundo se transmitirán 9600 caracteres). Por defecto, los *sketches* no envían ni reciben datos al monitor serie, hay que programarlo.

La **consola de mensajes** de la parte inferior del IDE Arduino **informa,** durante la compilación, **de los posibles errores de sintaxis** en el código del *sketch*. Además, permiten visualizar en tiempo real el estado de **los procesos de grabación** en el disco duro de los "*.ino", los procesos **de compilación** del *sketch*, **y** los procesos **de carga en el microcontrolador**. Cada vez que se realiza correctamente la compilación de un *sketch*, aparece en la consola de mensajes el **tamaño que ocupará el *sketch*** en la Flash del microcontrolador, lo cual puede ser interesante para comprobar que el sketch no supera el tamaño máximo permitido (de 32[KB] en la Flash de Arduino UNO), y si lo superase, poder modificar adecuadamente el código del *sketch*.

La barra de estado muestra, a la izquierda, el número de línea del sketch actual donde en donde tenemos situado el cursor, el tipo de placa Arduino y el puerto serie del computador utilizado en ese momento.

Estructura general de un sketch Arduino

S*ketch* se denomina a cualquier programa diseñado o desarrollado para ejecutarse sobre una placa Arduino. Principalmente se divide en tres secciones (comunes a cualquier *sketch)*. La parte superior del sketch, está reservada para importar las librerías de las que se vaya a hacer uso, definir las constantes del programa, y declarar las variables globales. A continuación, es imprescindible utilizar la función `void setup()`; el código escrito dentro de esta función, delimitada por llaves, será ejecutado una única vez, cada vez que se arranque la placa. Incluirá, por tanto, las configuraciones iniciales de la tarjeta Arduino. Finalmente, es necesario programar la función `void loop()`; esta función, también delimitada por llaves, será ejecutada inmediatamente después de la función `void setup()` y de forma ordenada y continua, como si de un bucle se tratase, hasta que apague la placa. Ha de incluir las instrucciones que se desean ejecutar permanentemente en la tarjeta Arduino y que integrarán el programa en sí.

Implementación del sketch, imagen 2

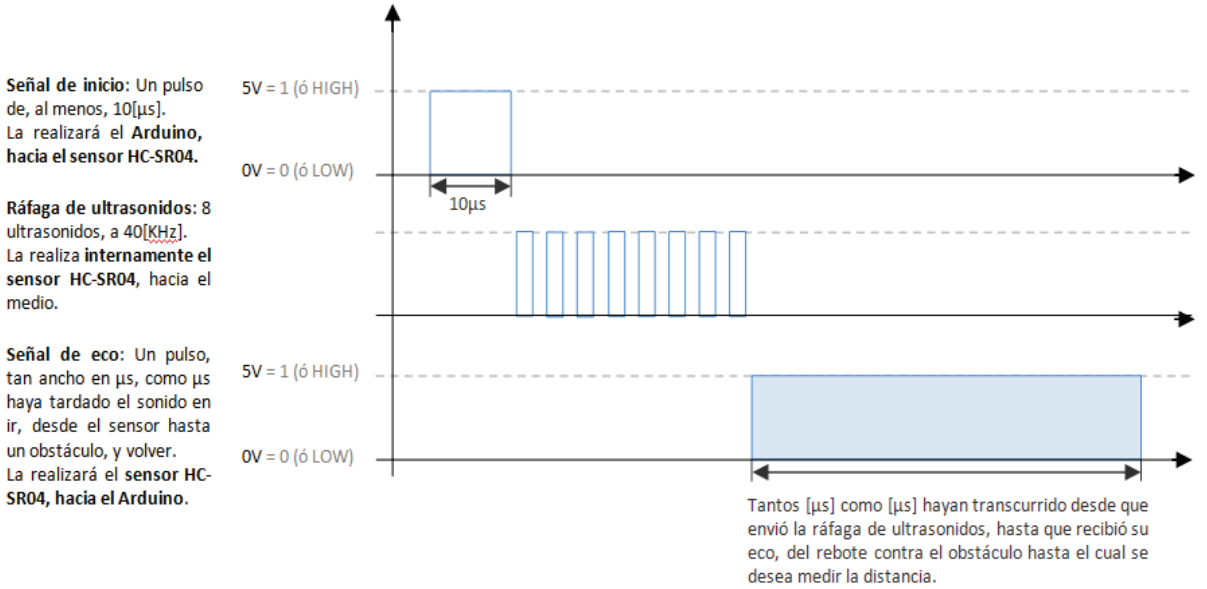
Código del programa

Para la medición de distancias mediante un sensor de ultrasonidos, se han de tratar las señales, para poder procesarlas y así, obtener como resultado, una medición, lo más exacta posible, de distancias a objetos. En las próximas líneas se explica de forma más extendida su funcionamiento.

Funcionamiento interno del sensor

Como ya se ha comentado el sensor HC-SR04 calcula la diferencia de tiempo que transcurre entre que transmite y recibe una serie de pulsos que él mismo envía y captura. A partir de este sistema de medición de tiempo, nos permite, calcular la distancia a un objeto.

Para entender mejor el funcionamiento interno del sensor y según el *datasheet* del sensor HC-SR04, en siguiente cronograma, se puede ver la representación de los pulsos con los que trabaja el sensor.



Implementación del sketch, imagen 3

De esta forma, y una vez obtenido el tiempo que ha medido el sensor, se puede calcular la distancia hasta un obstáculo, mediante la fórmula: **Distancia [cm] = ½ (Ancho de pulso de la señal de eco [µs])*(Velocidad del sonido [m/s]),** a partir de la cual se desarrolla el programa (*sketch*) con el que se implementará la medición de distancias en este proyecto.

Por otra parte, el sensor, como Arduino funciona a nivel TTL. Esto quiere decir que, a 0[V], o nivel bajo (LOW), el sensor HC-SR04 está en reposo; y a 5[V], o nivel alto (HIGH) se encuentra en funcionamiento. Además, sabiendo que el circuito consume la siguiente energía tal que Potencia[W] = Tensión[V] * Intensidad[mA], a través de un polímetro[91], se podría medir el consumo del circuito.

Comunicación de Arduino UNO con el Sensor HC-SR04

Arduino tiene que enviar al sensor una señal para que el sensor entre en funcionamiento. Para ello, y según la hoja de especificaciones del sensor HC-SR04, se ha de enviar un **pulso de**, al menos, **10[µs]** desde el pin TRIG.

El **pin TRIG** es la E/S digital que funciona como *trigger* o disparador. Este pin es a través del cual Arduino ha de enviar al sensor la señal de comienzo (el pulso de 10[µs]) para indicar al sensor que empiece a funcionar. Es por lo tanto un pin de **salida** (output) **digital de Arduino** y un pin de entrada digital del sensor. Mediante código se configura este pin de la siguiente forma:

```
pinMode(PIN_TRIG,OUTPUT);
```

A través del pin TRIG se ha de que **enviar información** al sensor, es decir habrá que **escribir** a través de ese pin. La información que se ha de enviar, es **un pulso**, por lo tanto, habrá que escribir un pulso a nivel alto (**HIGH**), es decir un **1 lógico**, que a su vez corresponde con **5[V]** de tensión (1 corresponde a 5[V] porque Arduino y el sensor trabajan a nivel TTL). Mediante código se realiza así:

```
digitalWrite(PIN_TRIG,HIGH);
```

Ya que el pulso del pin configurado como TRIG ha de ser de, al menos, 10[µs], este será el tiempo que se ha de mantener el pulso a nivel alto. Se puede esperar con esta instrucción:

```
delayMicroseconds(10);
```

A continuación, se terminará dicho pulso. Para terminar el pulso, ha de ponerse a nivel bajo (LOW), es decir un **0 lógico**, que a su vez corresponde con **5[V]** de tensión (0 corresponde a 0[V] TTL).

```
digitalWrite(PIN_TRIG,LOW);
```

Como la comunicación serie en Arduino se realiza de forma asíncrona, es necesario indicar la velocidad a la que queremos que se comunique con el sensor y con el PC. Se ha optado por una velocidad de transmisión información de 115200 baudios, que permite que los datos aparezcan más

[91] Un polímetro o multímetro digital es simplemente un instrumento que sirve para medir magnitudes, entre otras, relacionadas con la ley de Ohm, V[V] = I[A]*R[Ω], es decir, el voltaje entre dos puntos de un circuito, V; la intensidad de la corriente que fluye a través del circuito, I; o la resistencia que ofrece algún componente, R.

lentamente en el monitor serie del IDE, y así, mayor comodidad a la hora de realizar las pruebas. Cada vez que se envíen datos desde la placa al monitor serial, parpadeará el led TX de la placa.

```
Serial.begin(115200);
```

Comportamiento del Sensor HC-SR04 en el medio

En cuanto el sensor detecte que Arduino le ha enviado un pulso, es decir, cuando haya **terminado el pulso de 10[µs], el sensor entrará en funcionamiento**. El sensor comienza a funcionar enviando, a través del transductor cilíndrico (a modo de altavoz emisor) que incorpora, una ráfaga de 8 pulsos de ultrasonidos a 40[KHz] (lo cual se indica en el *datasheet* del sensor). Para entendernos, es como si enviase, 8 ultrasonidos (sonidos a una frecuencia que el oído humano no puede percibir).

Una vez emitidos los 8 pulsos de 40[KHz], el sensor **espera** un tiempo hasta que le llegue, a través del transductor cilíndrico (a modo de micrófono receptor), el eco del ultrasonido que ha generado. **Desde que envía** el primer pulso de la ráfaga de ultrasonidos, **hasta que le llega** el primer **eco** proveniente del rebote de los ultrasonidos al chocar con un obstáculo, **se inicia** en el sensor **un conteo del tiempo en microsegundos.**

Este tiempo que ha cronometrado el sensor servirá, mediante un sistema de tratamiento de esta señal, para calcular la distancia al objeto. Habrá que enviar dicho tiempo a la placa Arduino (al su microcontrolador) para que lo interprete.

Comunicación del Sensor HC-SR04 con Arduino UNO

El sensor, para indicar a la placa Arduino que ha recibido el eco de la ráfaga de ultrasonidos que él mismo envió**, tiene que enviar a la placa Arduino una señal**, en forma de pulso, a través del pin ECHO del sensor.

El **pin ECHO** es la E/S digital que avisa (con un pulso) a la placa Arduino de que enviado y recibido el eco (o *echo*) de la ráfaga de ultrasonidos. Se trata por tanto, de un pin de salida digital del sensor y de entrada (intput) digital de la placa Arduino.

```
pinMode(PIN_TRIG,OUTPUT);
```

Este **pin** sirve **para calcular cuánto tiempo** tarda la ráfaga de ultrasonidos en ir (desde el sensor, al obstáculo) **y volver** (desde el obstáculo, al sensor). Dicho tiempo, se corresponderá al

tiempo que permanece el pulso que ha enviado el sensor a la placa Arduino, a nivel alto (HIGH). Tras recibir el eco, el pulso se pondrá de nuevo a nivel bajo.

La función `pulseIn(pin,nivel);` del lenguaje de programación de Arduino, indica la anchura de un pulso (en este caso, la anchura del pulso a nivel alto, que ha enviado el pin ECHO), en microsegundos; es decir, el tiempo que ha estado el pulso a nivel alto (HIGH).

```
tiempo = pulseIn(PIN_ECHO,HIGH);
```

En las hojas de especificaciones del sensor se recomienda utilizar un tiempo de espera de unos 50 o 60 [µs] para asegurar que los pulsos del pin ECHO no comienzan antes de que se haya enviado la ráfaga de 8 pulsos de ultrasonidos completamente.

```
delay(60);
```

Comportamiento de Arduino UNO en el medio

La placa Arduino a través de su microcontrolador, tendrá que estimar la distancia a un punto (obstáculo de la ráfaga de ultrasonidos). Suponiendo un movimiento rectilíneo uniforme de la ráfaga de ultrasonidos y sirviéndose de la fórmula de la velocidad, que es $\boldsymbol{v = d/t}$, donde *v* es la velocidad, *d* la distancia, y *t* el tiempo, se podrá calcular la distancia hasta un obstáculo.

A velocidad constante en un tiempo determinado, la distancia sería: $d = v * t$. Aplicado a este caso, la formula quedaría de la siguiente forma:

- $d = distancia\ a\ un\ objeto$. Será la distancia que hay desde el sensor hasta el obstáculo (u objeto) que haga que rebote la ráfaga de ultrasonidos. Se mide en centímetros, [cm].

- $v = velocidad\ de\ la\ ráfaga\ de\ ultrasonidos\ enviada$. La velocidad de los ultrasonidos es la misma que la velocidad del sonido: 340[m/s] a 25[°C] de temperatura, ó 343[m/s] a 20[°C].

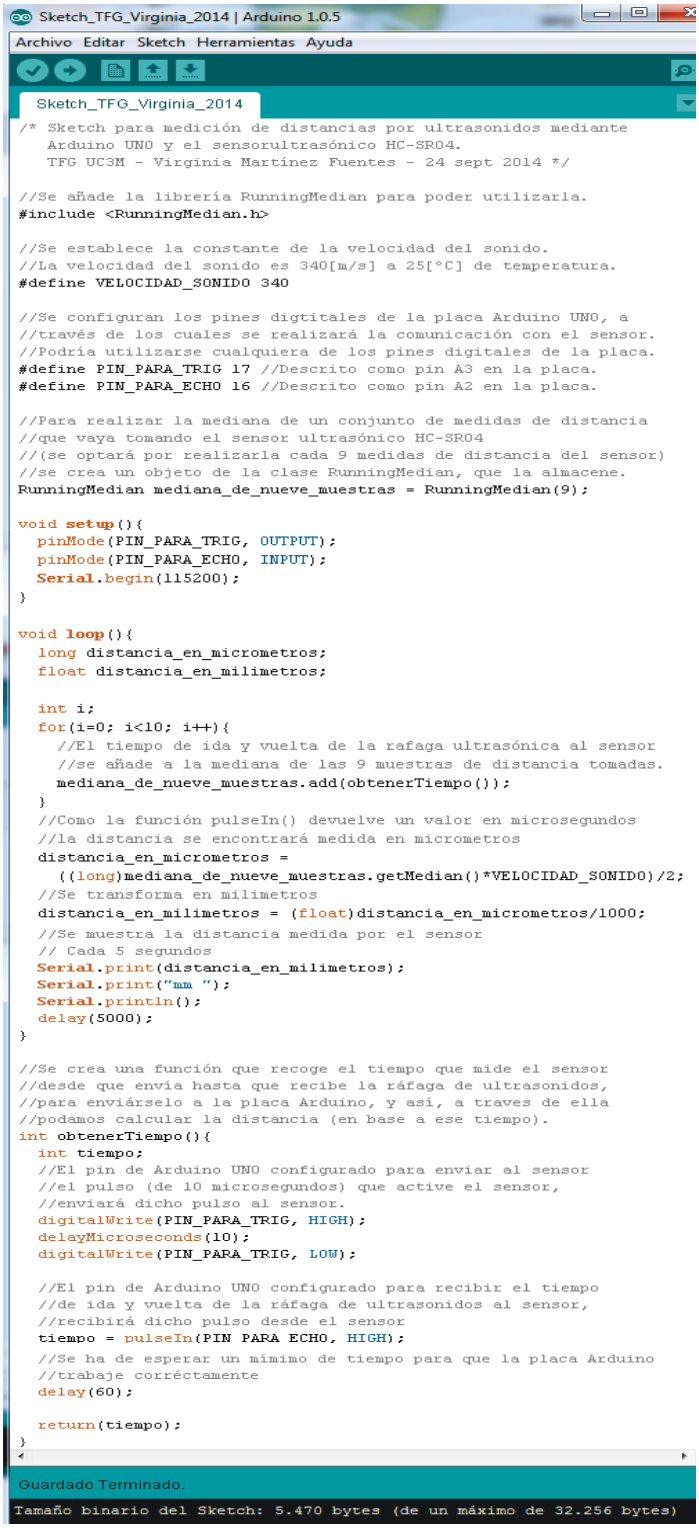
- $t = ½\ del\ tiempo\ que\ ha\ estado\ el\ pulso\ del\ pin\ echo\ a\ nivel\ alto$. El tiempo que ha tardado la ráfaga de ultrasonidos en llegar a un obstáculo, es la mitad (½) del tiempo que ha tardado la ráfaga de ultrasonidos en ir y volver al sensor (que se ha medido, midiendo la duración del pulso del pin ECHO). Se mide en microsegundos, [µs].

Además, se ha utilizado la librería `RunningMedian.h` (que se explica posteriormente, en el subapartado "Explicación de las bibliotecas utilizadas") para hallar la **mediana de cada x conjunto de muestras** (se ha decidido que sea cada 9, como se podría haber tomado otro número de las mismas) **de distancia tomadas por el sensor**, de forma que la precisión de la medición sea más exacta.

```
RunningMedian(9);
```

Código del sketch final

La imagen que se expone a continuación, muestra el *sketch* definitivo a partir del cual se han realizado la medición de distancias con Arduino UNO y el sensor HC-SR04, y las pruebas al respecto.



Implementación del sketch, imagen 4

Explicación de las funciones de Arduino utilizadas

Para la gestión de las entradas y salidas digitales de la placa se han utilizado funciones como "`pinMode(pin,valor);`", "`digitalWrite(pin,valor);`" y "`pulseIn(pin,nivel);`". Para la comunicación serie con la placa se ha hecho uso de "`Serial.begin();`". Para enviar datos desde la placa al exterior, de la función "`Serial.print();`". Y para gestión de tiempos, de "`delay();`". Todas estas funciones[92], facilitadas por el lenguaje de programación Arduino, se explican con más detalle a continuación.

`pinMode(pin,valor);`

Configura el pin digital, cuyo número se especifica como primer parámetro, como entrada o salida de corriente, según se especifique en su segundo parámetro (`INPUT`, si el pin se quiere definir como entrada, o `OUTPUT`, si se quiere definir como salida). Esta función no tiene valor de retorno.

Esta función es útil en cuanto a que los pines digitales, en principio, pueden actuar como entrada o como salida, pero se han de definir en el sketch, para que actúen de una u otra forma. Esta función suele ir dentro de la estructura `setup();`.

Si el pin a configurar se quiere utilizar como entrada, se puede activar la resistencia "*pull-up*" (de 20[KΩ]) que incorporan todos los pines digitales (mediante el `INPUT_PULLUP`, en vez de `INPUT`, ya que la constante `INPUT` desactiva estas resistencias "*pull-up*"). Si un pin de entrada tiene su resistencia "*pull-up*" desactivada, cuando el pin no esté conectado a nada puede recibir el ruido eléctrico que le rodea y provocar distorsiones en los valores de entrada obtenidos, que variarían de manera aleatoria. Otra manera de activar la resistencia "*pull-up*" es utilizar la constante `INPUT` junto con la función `digitalWrite();`. Y otra forma de conseguir el mismo resultado práctico es utilizar un resistencia "*pull-down*" externa (resistencia que conecta ese pin a tierra) de unos 10[KΩ].

`Serial.begin(velocidad);`

Esta función abre el canal de comunicación serie, por lo que es imprescindible configurarla antes de cualquier transmisión de este tipo (se suele incluir dentro del bloque del `setup()`). Su parámetro, obligatorio y de tipo `long`, indica la velocidad (en baudios, entendidos normalmente como bits por segundo) de transmisión de información. Esta función no tiene valor de retorno.

[92] Tanto estas funciones, como el resto de funciones del lenguaje Arduino se encuentran explicadas en el sitio web de Arduino (Enlace:). Además, también pueden ser consultadas en el Anexo I de este documento.

Es importante, para que la comunicación (transmisión de información entre el computador y la placa) se sincronice correctamente, que la **velocidad indicada en el parámetro** de esta función**, coincida con la velocidad del monitor serie** ("o serial") **del IDE Arduino**.

```
digitalWrite(pin,valor);
```

Esta función envía el valor especificado en el segundo parámetro (`HIGH` o `LOW`, ambas constantes de tipo `int`), al pin digital descrito (con su número de pin) como primer parámetro. Esta función no tiene valor de retorno.

Si el pin especificado como parámetro está ha sido configurado como **salida** (`OUTPUT`), y en el segundo parámetro de la función se pone un valor alto (la constante `HIGH`), este valor equivaldrá a una salida de hasta 40[mA], operando a 5[V] (porque ese es el voltaje de trabajo de Arduino). Si el valor alto (`HIGH`, es decir, 5[V]) que ofrece un pin de salida digital no es suficiente como para alimentar a los componentes del circuito (porque sean de alto consumo, como motores o matrices de leds), se tendrá utilizar más circuitería extra que lo permita, para evitar daños en el pin por sobrecalentamiento. Y por el contrario, puede ocurrir que ese valor de voltaje deba ser reducido, y haya que emplear un divisor de tensión, para adecuarlo al voltaje de trabajo del componente.

Si el pin especificado como parámetro ha sido configurado como **entrada** (`INPUT`), y en el segundo parámetro de la función se pone un valor alto (la constante `HIGH`), este valor equivaldría a activar la resistencia interna "*pull-up*" en ese momento, y sería lo mismo que haber configurado el pin directamente como `INPUT_PULLUP`. Y si en el segundo parámetro se pone un valor bajo (`LOW`), este valor equivaldría a desactivar la resistencia interna "*pull-up*" de nuevo.

```
pulseIn(pin,nivel);
```

Detiene la ejecución del sketch y espera a recibir, en el pin de entrada señalado como primer parámetro, la señal especificada en el segundo parámetro (`HIGH` o `LOW`). Tras recibir la señal, cuenta el tiempo que dura dicha señal (pulso) hasta que cambie de estado, y lo devuelve como un valor de tipo `long`, en microsegundos.

De forma opcional, se puede añadir a esta función, un tercer parámetro (de tipo `unsigned long`) que representa el tiempo máximo de espera en microsegundos. Si la señal no llega una vez transcurrido ese tiempo, la función devolverá un 0 y continuará la ejecución del *sketch*. Si este parámetro no se especifica, el tiempo de espera por defecto es de 1 minuto ($6*10^7$[µs]).

Se recomienda utilizar esta función para rangos de valores de entre aproximadamente 10[µs] y 3[min], ya que, para pulsos de más duración que esa, la precisión puede ser errónea.

```
Serial.print(valor);
```

Esta función envía el dato especificado como parámetro, desde el microcontrolador de la placa hacia el exterior, a través del canal serie. La transmisión de información con esta función se realiza de forma asíncrona. Esta función devuelve el número de bytes (caracteres) enviados (un dato de tipo byte).

Si el parámetro es una variable, puede ser de cualquier tipo, si es un carácter, se ha de escribir entre comillas simples y si se trata de una cadena, entre comillas dobles. Por otra parte, si el parámetro es una variable de tipo entero, se puede añadir a esta función un segundo parámetro opcional con una constante que indique su representación (`BIN`, `HEX`, `DEC`) que, por defecto, es decimal. Si el parámetro es una variable de tipo decimal, se puede añadir a esta función un segundo parámetro opcional con una el número de decimales a utilizar que, por defecto, son dos.

```
delay(milisegundos);
```

Esta función pausa la ejecución del *sketch* durante la cantidad de milisegundos especificados como parámetro, de tipo `unsigned long`. No tiene valor de retorno.

Explicación de las bibliotecas *(libraries)* utilizadas

`RunningMedian.h`

Se trata de una clase ya creada que podemos utilizar con Arduino para hallar la mediana de un conjunto de valores. La mediana es el valor que se encuentra en el medio de un conjunto de valores ordenados (si se trata de un conjunto de números par, será el promedio de los dos valores que se encuentren en medio).

Para utilizar esta librería basta con añadirla a la carpeta donde se encuentra descargado en *software* Arduino en nuestro ordenador: Arduino>libraries. Para ello, se puede crear una nueva carpeta Arduino>libraries>RunningMedian la cual ha de contener dos archivos: RunningMedian.ccp (que contiene el cuerpo de las funciones) y RunningMedian.h (el archivo de cabecera en donde principalmente se definen los prototipos de las funciones). El código de ambos archivos que forman

la librería RunningMedian se pueden obtener del sitio web de Arduino[93]. Una vez agregada esta nueva librería, ya se puede hacer uso de ella. Al tratarse de una librería, ha de encontrarse en la parte más superior del *sketch* (antes de cualquier otra línea de código) de la forma:

```
#include <RunningMedian.h>
```

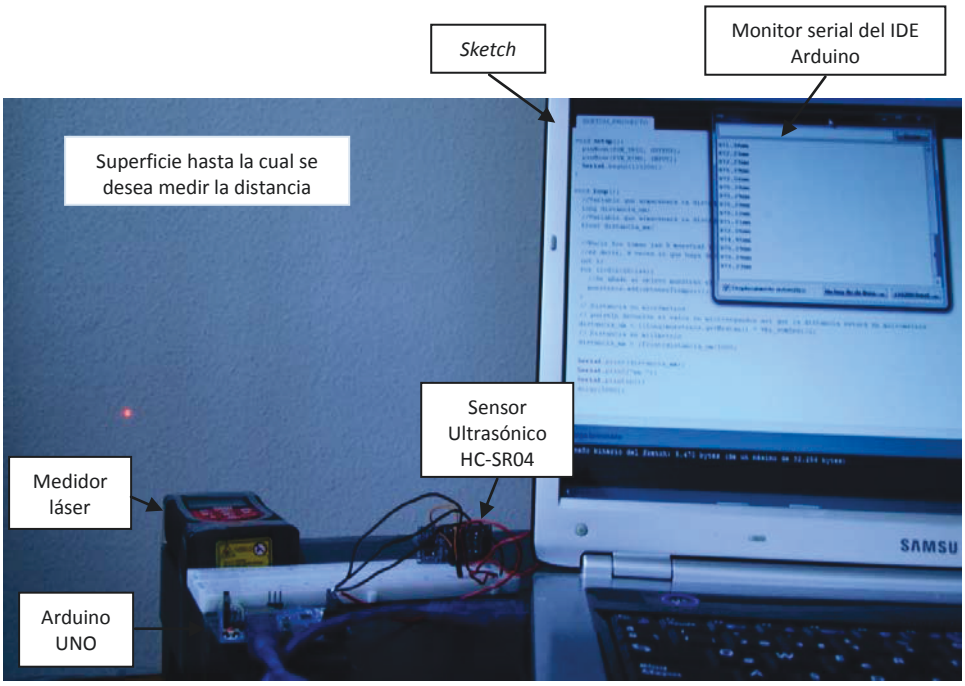
Como alternativa a esta librería, se podría haber utilizado la librería "NewPing" o la librería "SR04". Pero se ha decidido utilizar la librería "RunningMedian" puesto que las librerías "NewPing" y "SR04" sólo alcanzaban la precisión a nivel de centímetros.

[93] En una nueva carpeta dentro de Arduino>libraries, se ha de crear un archivo RunningMedian.h y un archivo RunningMedian.ccp formado por el código correspondiente en cada una, tal y como se muestra en el enlace: http://playground.arduino.cc/Main/RunningMedian.

EVALUACIÓN Y RESULTADOS

Pruebas

Para la realización de las pruebas del sistema (tal y como se muestra en la próxima imagen), se ha dispuesto sobre una mesa el **conjunto** que forman la placa **Arduino UNO, el sensor ultrasónico HC-SR04 y la placa de prototipos**, **junto a un medidor láser** de distancia[94], de alta precisión, similar al que se utiliza para la medición de distancias en el desempeño de actividades relacionadas con la arquitectura técnica. De esta forma, se puede estudiar más certeramente el grado de error que se obtiene al medir las distancias mediante Arduino UNO y el HC-SR04, comparando los resultados con la distancia medida por dicho láser. La placa de prototipado no es un elemento indispensable en este circuito, pero es útil en cuanto a que permite que el sensor permanezca en una posición estable mientras se toman las medidas, cuyos, valores se visualizan en el IDE Arduino, al conectar un cable USB a la placa Arduino UNO y ejecutar el *sketch* correspondiente.



Pruebas, imagen 1

[94] A la hora de realizar las pruebas, se ha utilizado un instrumento medidor con sensor láser, del fabricante "Skil" (modelo 0530 AA), que ha sido proporcionado por el departamento de Informática de la Universidad. Si se desean consultar las características técnicas y especificaciones de este medidor láser, puede consultarse su sitio web: http://www.skileurope.com/es/es/diyocs/herramientas/1208/508/medicion-de-distancias/medidor-de-distancias-por-laser/.

A continuación, se muestran gráficamente los resultados obtenidos en las diversas pruebas que se han realizado.

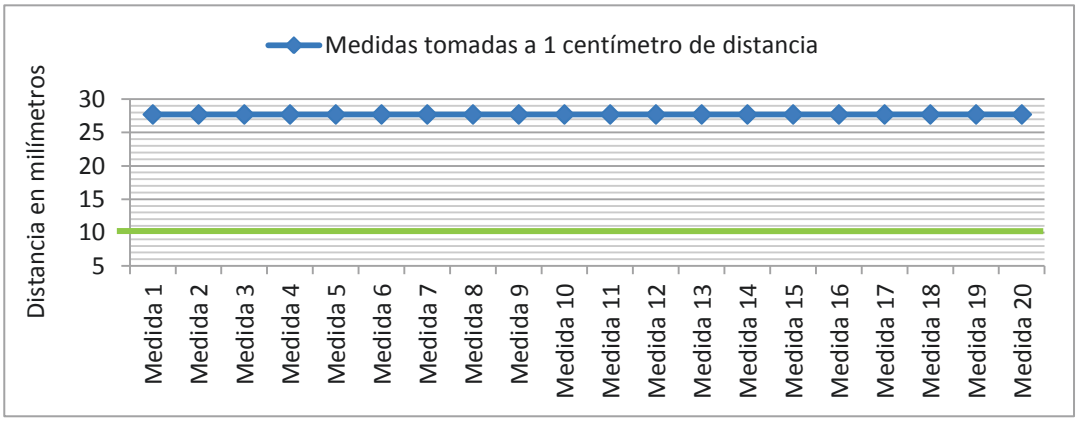
En el eje vertical (Y) de cada una de las gráficas se representa la distancia exacta a un punto de un objeto o persona, según el caso. Esta distancia se tomará como referencia para, comparándola con la distancia medida por el sensor, mostrar el grado de error que éste tiene asociado. Para saber la distancia real a la que se encuentra el obstáculo de los ultrasonidos en cada una de las pruebas, se ha utilizado el medidor láser de distancias (para las pruebas en distancias a partir de 20 centímetros, el mínimo que este instrumento puede proporcionar) o una regla o metro (en distancias inferiores a los 20[cm]). Se ha de considerar que siempre cabe la posibilidad de introducir errores de medición, fruto del factor humano en la experimentación.

En el eje horizontal (X), aparecen las medidas que va tomando el sensor HC-SR04 al mismo punto al que se tomo la distancia de referencia. En cada una de las gráficas, se han tomado veinte muestras consecutivas de medición del sensor, las cuales han sido mostradas en el monitor serie del IDE Arduino.

Prueba 1. Distancia a una superficie vertical (90° respecto al suelo).

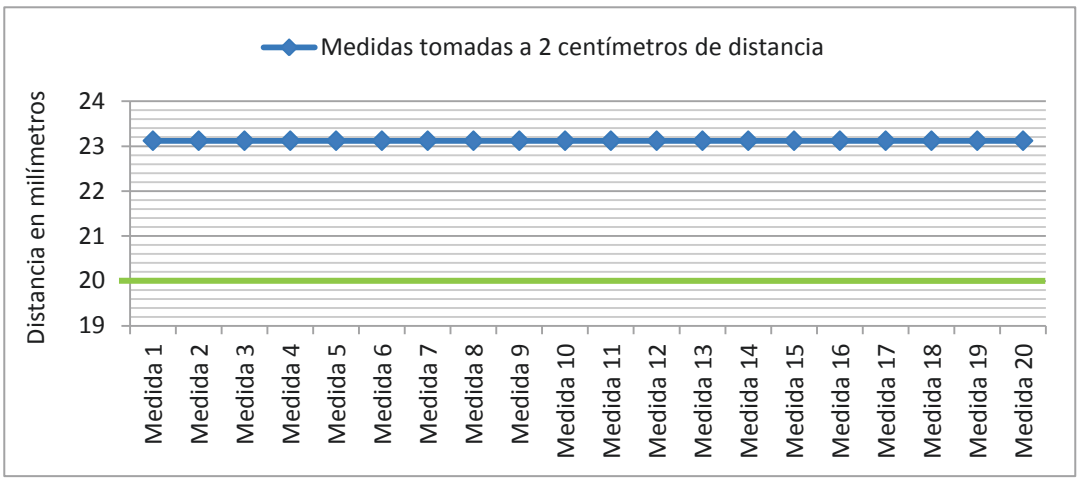
Las siguientes pruebas muestran los resultados de medición obtenidos gracias a la placa Arduino UNO junto al sensor HC-SR04, a partir del *sketch* explicado. Las medidas han sido tomadas desde el sensor, hasta una superficie vertical amplia, tal como una pared. Las pruebas a partir de 20[cm] de distancia han sido realizadas contra una superficie rugosa (una pared blanca con gotelé), mientras que las pruebas de entre 1 y 15[cm] de distancia se han realizado hacia una superficie lisa (una pared con azulejos marrones).

Prueba 1.1. Desde el sensor hasta una pared a 1[cm] de distancia.



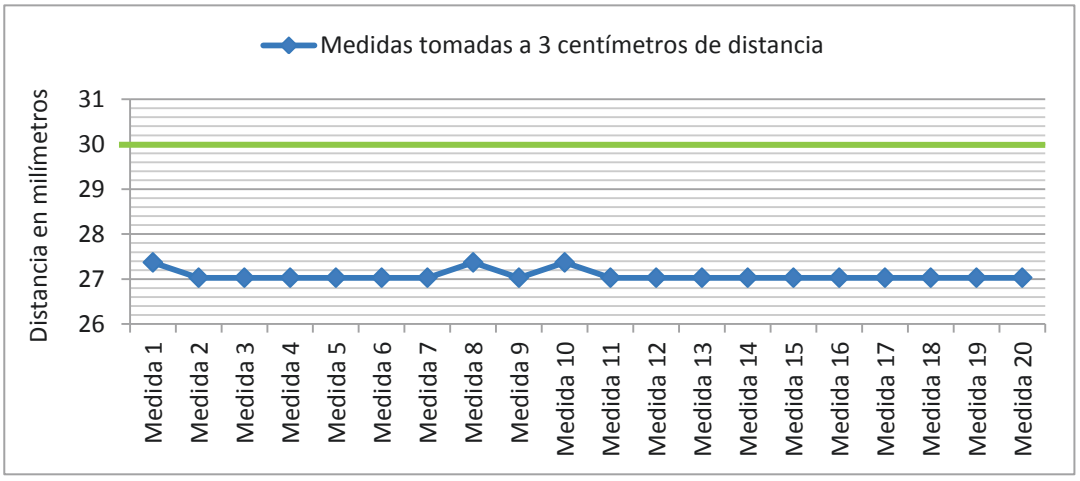
Prueba 1. 1

Prueba 1.2. Desde el sensor hasta una pared a 2[cm] de distancia.



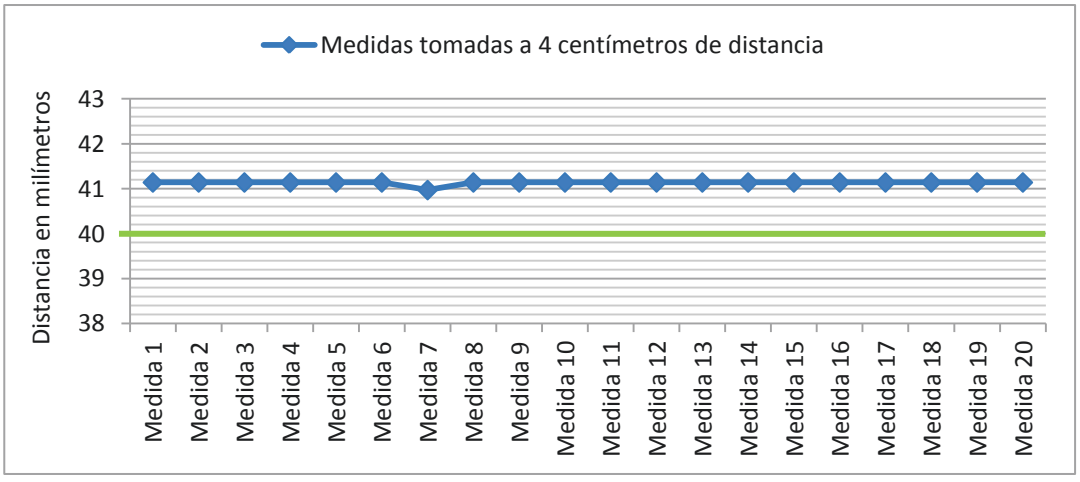
Prueba 1. 2

Prueba 1.3. Desde el sensor hasta una pared a 3[cm] de distancia.



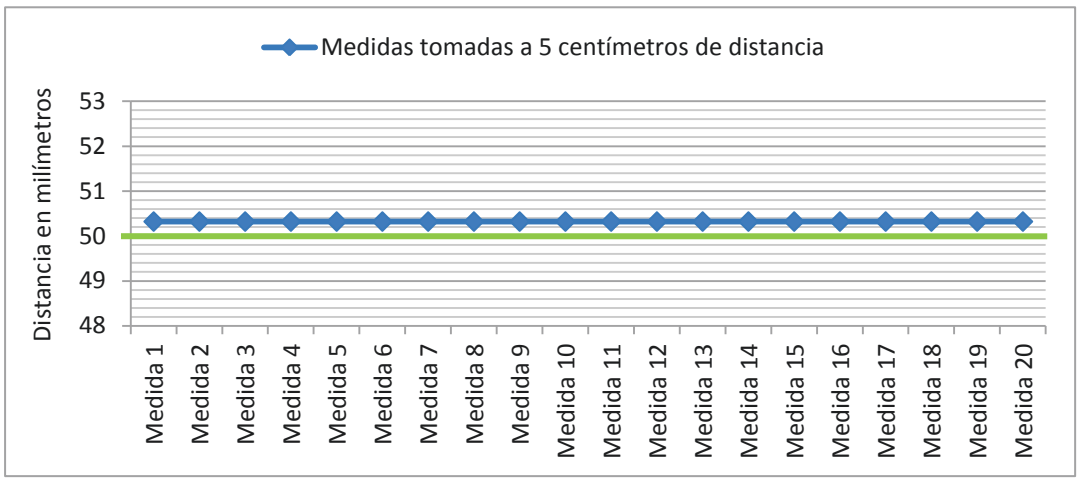
Prueba 1. 3

Prueba 1.4. Desde el sensor hasta una pared a 4[cm] de distancia.



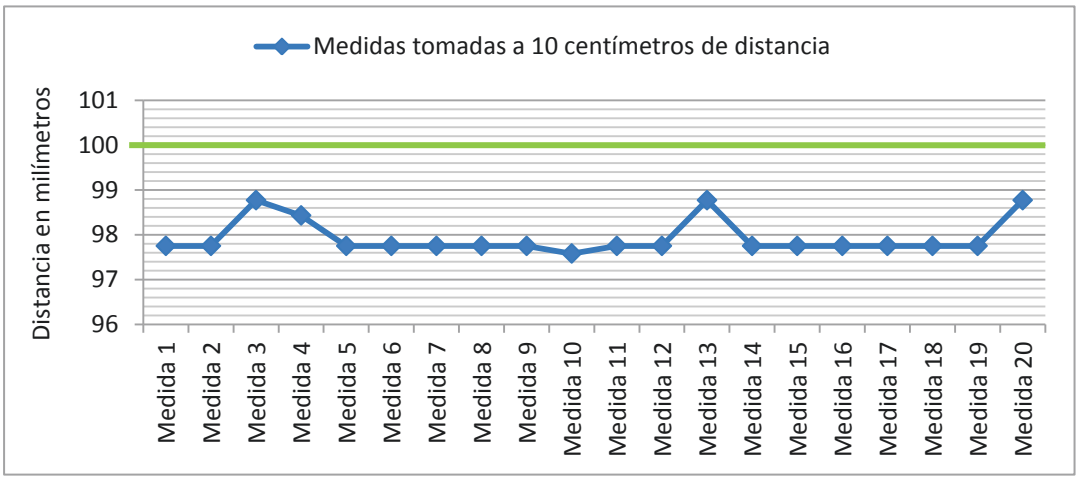
Prueba 1. 4

Prueba 1.5. Desde el sensor hasta una pared a 5[cm] de distancia.



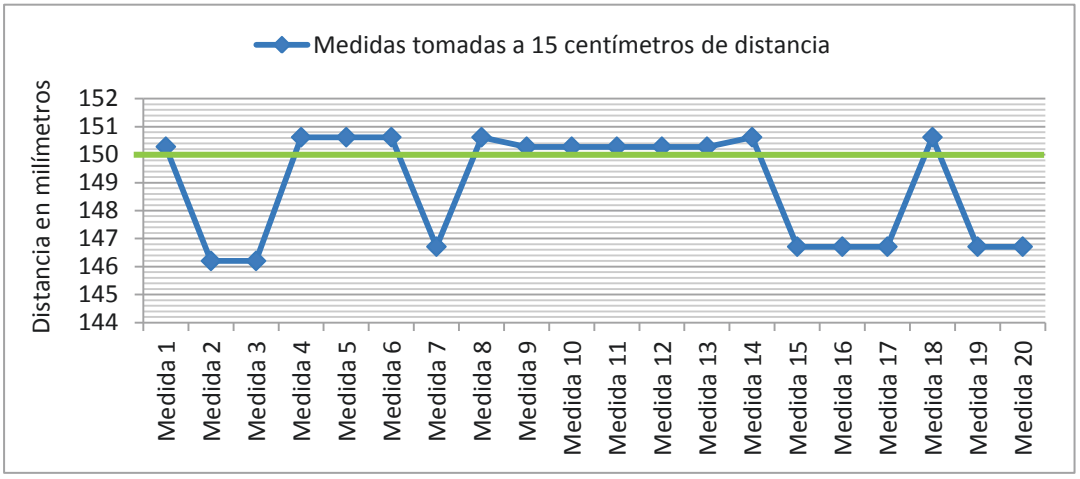
Prueba 1. 5

Prueba 1.6. Desde el sensor hasta una pared a 10[cm] de distancia.



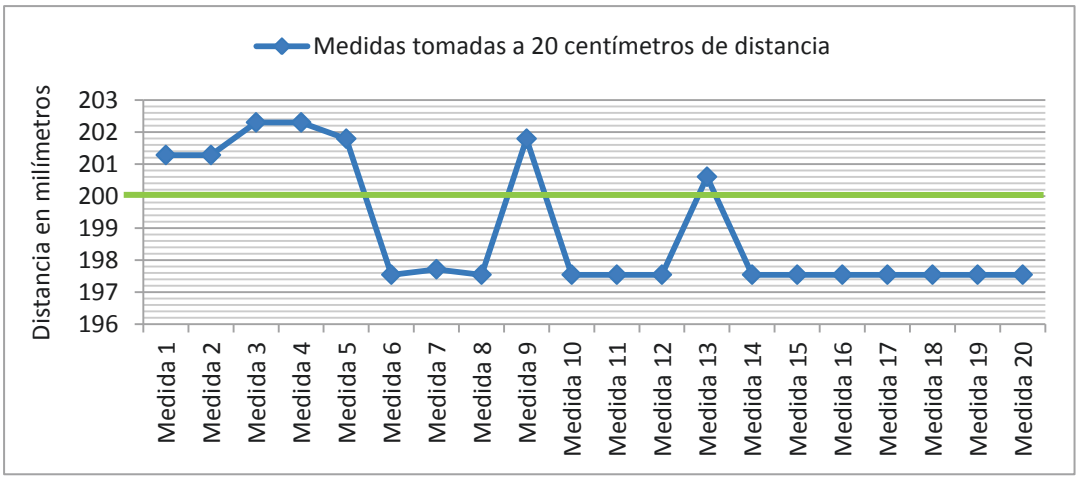
Prueba 1. 6

Prueba 1.7. Desde el sensor hasta una pared a 15[cm] de distancia.



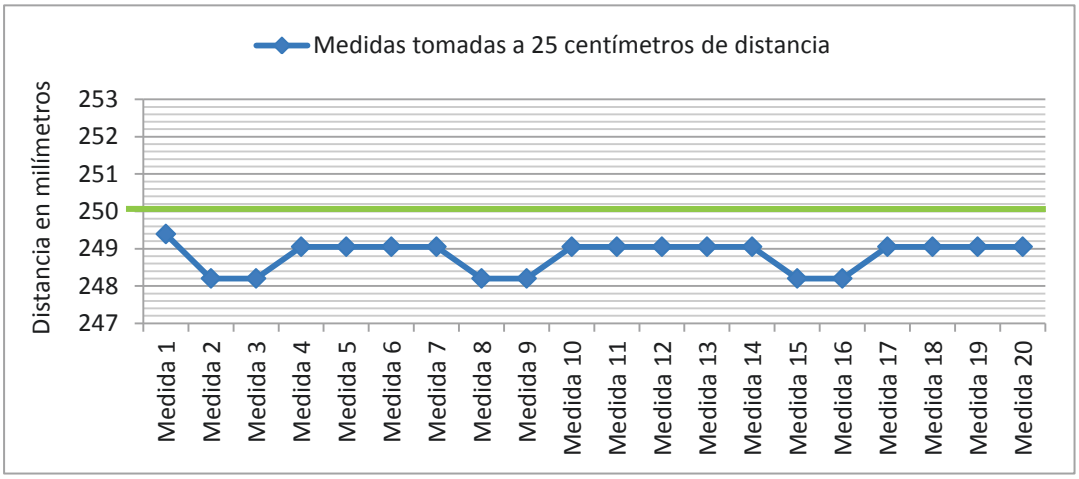
Prueba 1. 7

Prueba 1.8. Desde el sensor hasta una pared a 20[cm] de distancia.



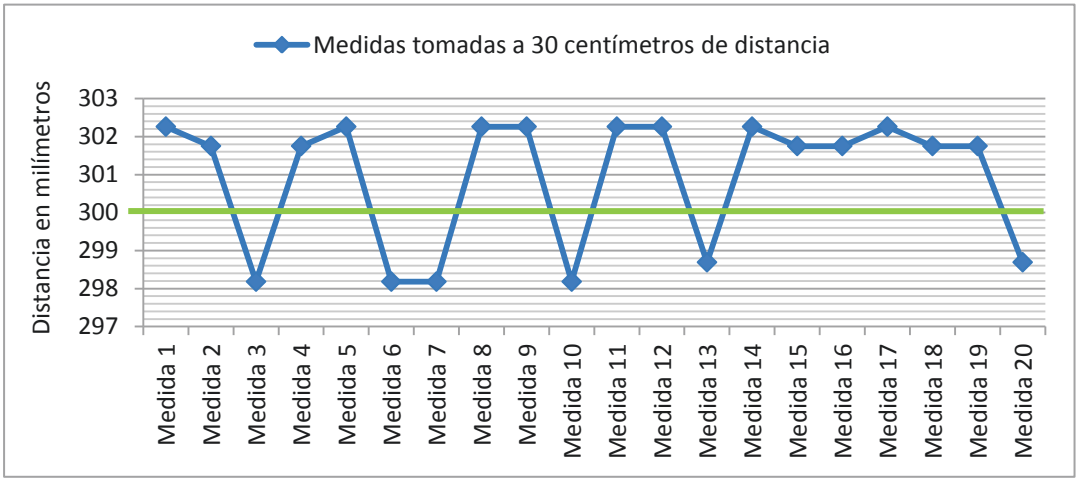
Prueba 1. 8

Prueba 1.9. Desde el sensor hasta una pared a 25[cm] de distancia.



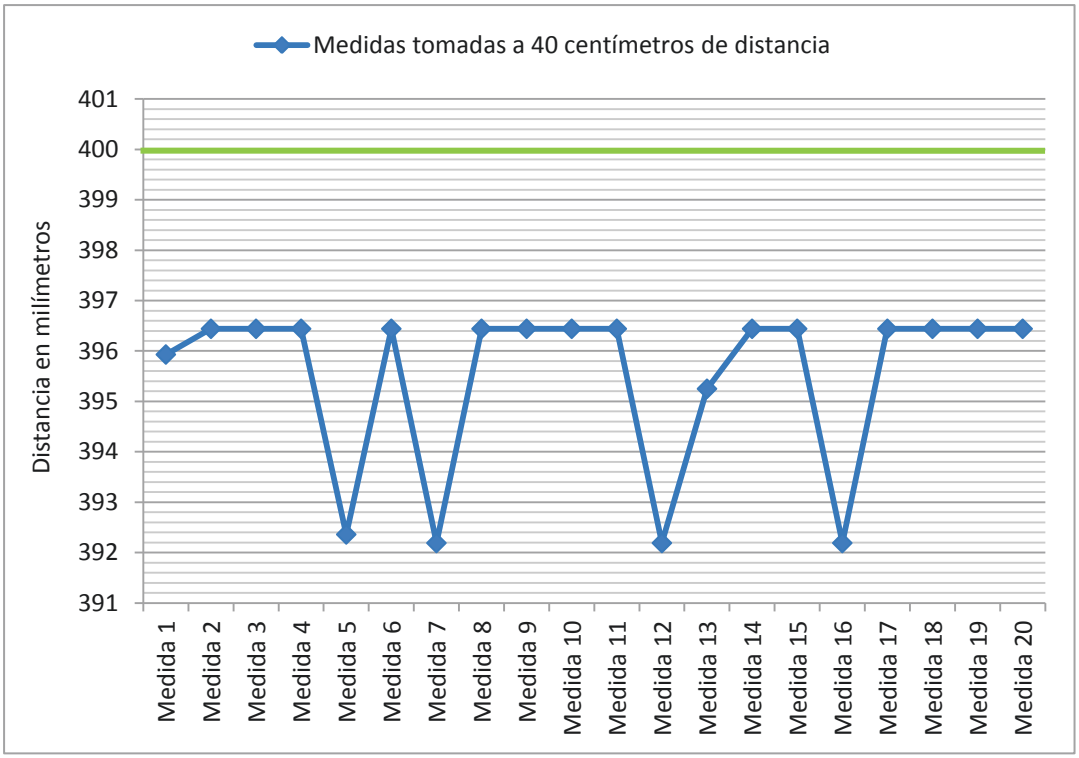
Prueba 1. 9

Prueba 1.10. Desde el sensor hasta una pared a 30[cm] de distancia.



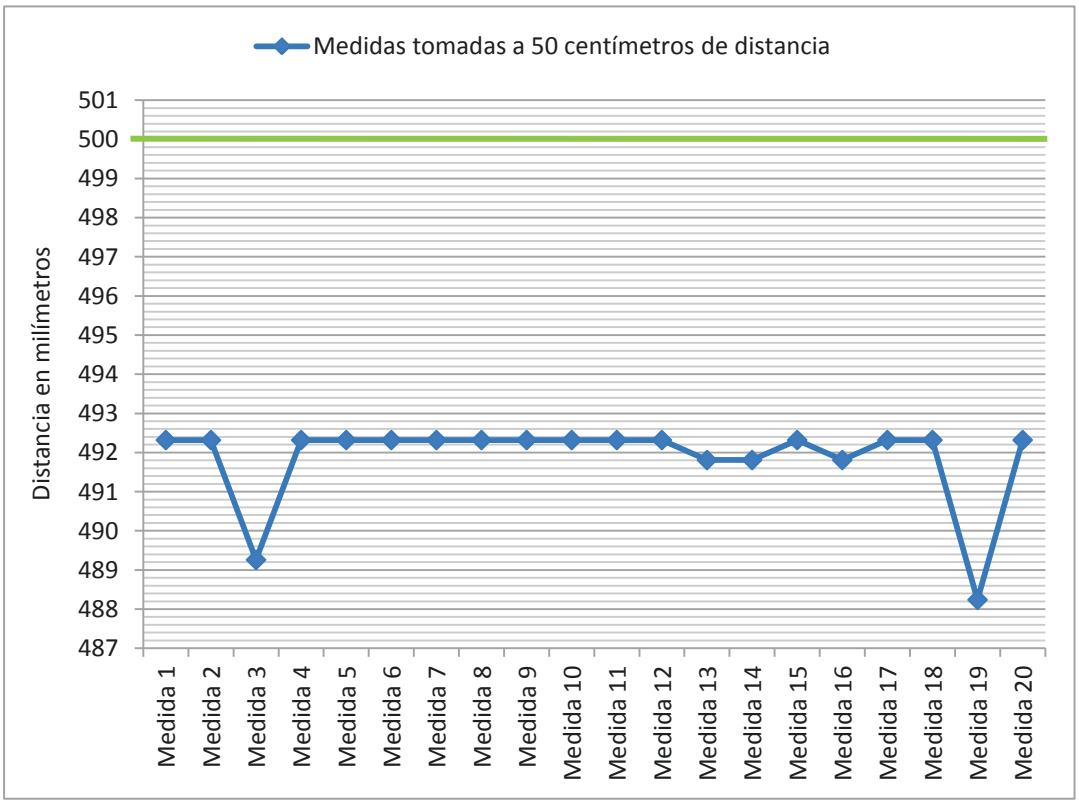
Prueba 1. 10

Prueba 1.11. Desde el sensor hasta una pared a 40[cm] de distancia.



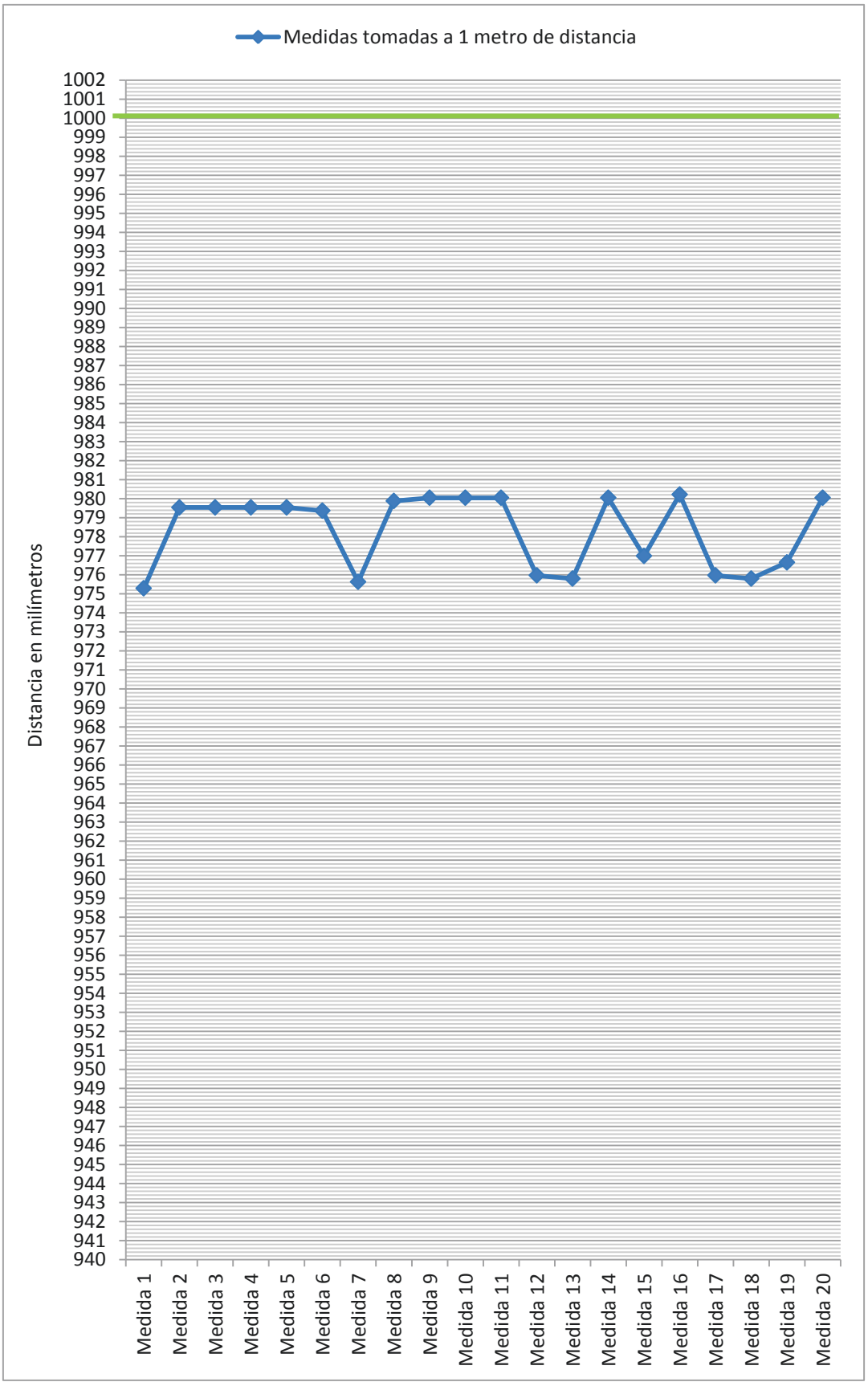
Prueba 1. 11

Prueba 1.12. Desde el sensor hasta una pared a 50[cm] de distancia.



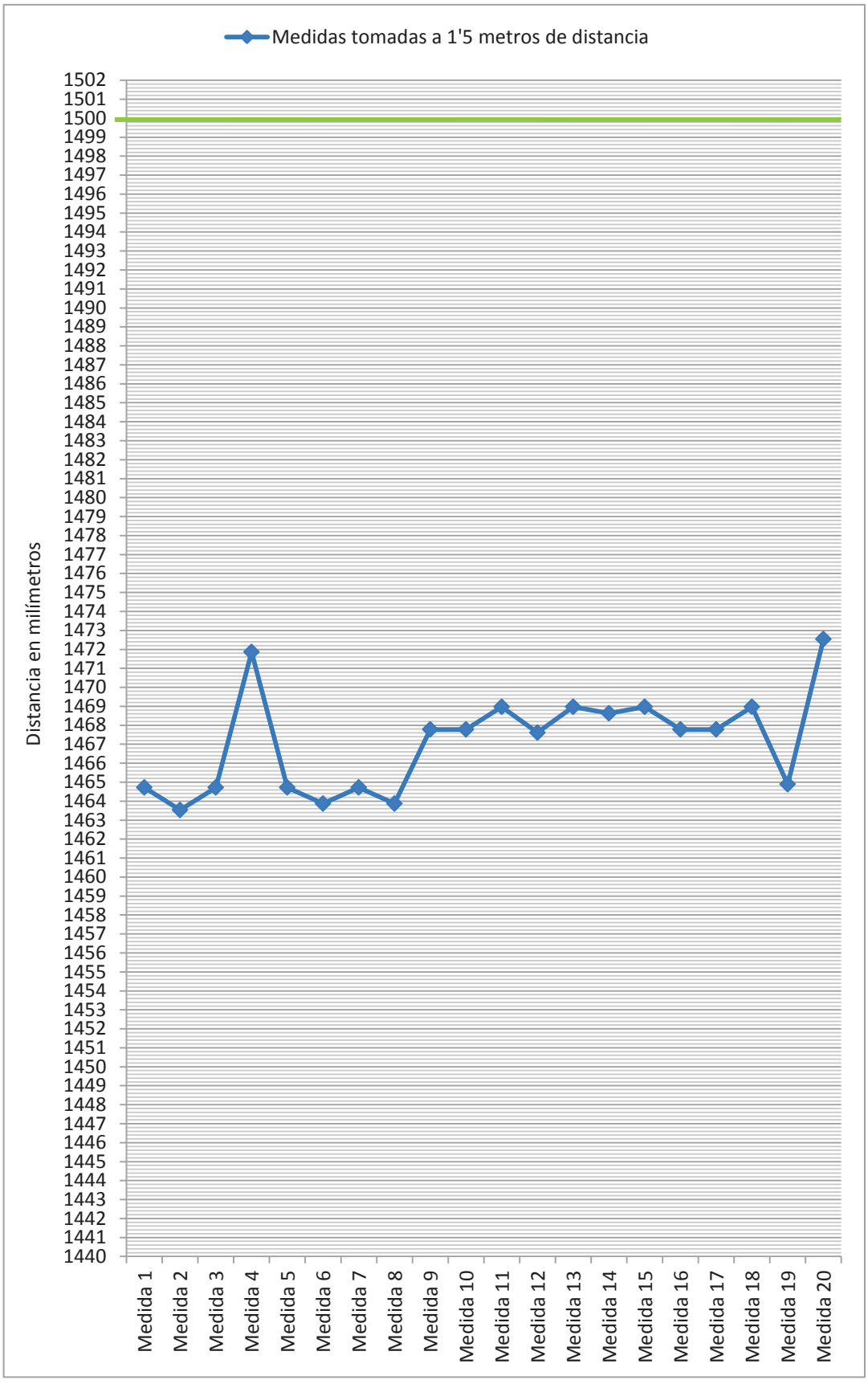
Prueba 1. 12

Prueba 1.13. Desde el sensor hasta una pared a 1[m] de distancia.



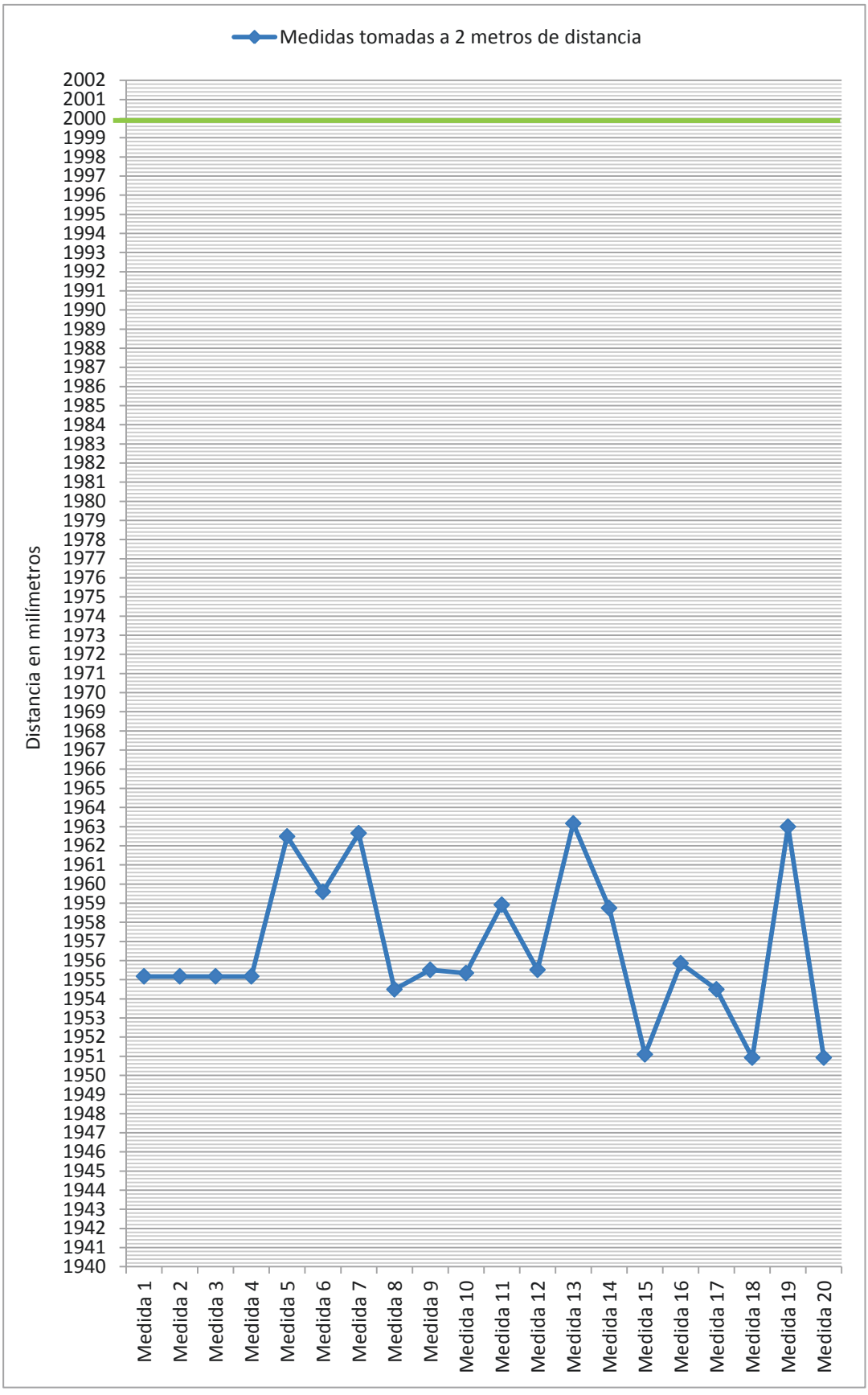
Prueba 1. 13

Prueba 1.14. Desde el sensor hasta una pared a 1'5[m] de distancia.



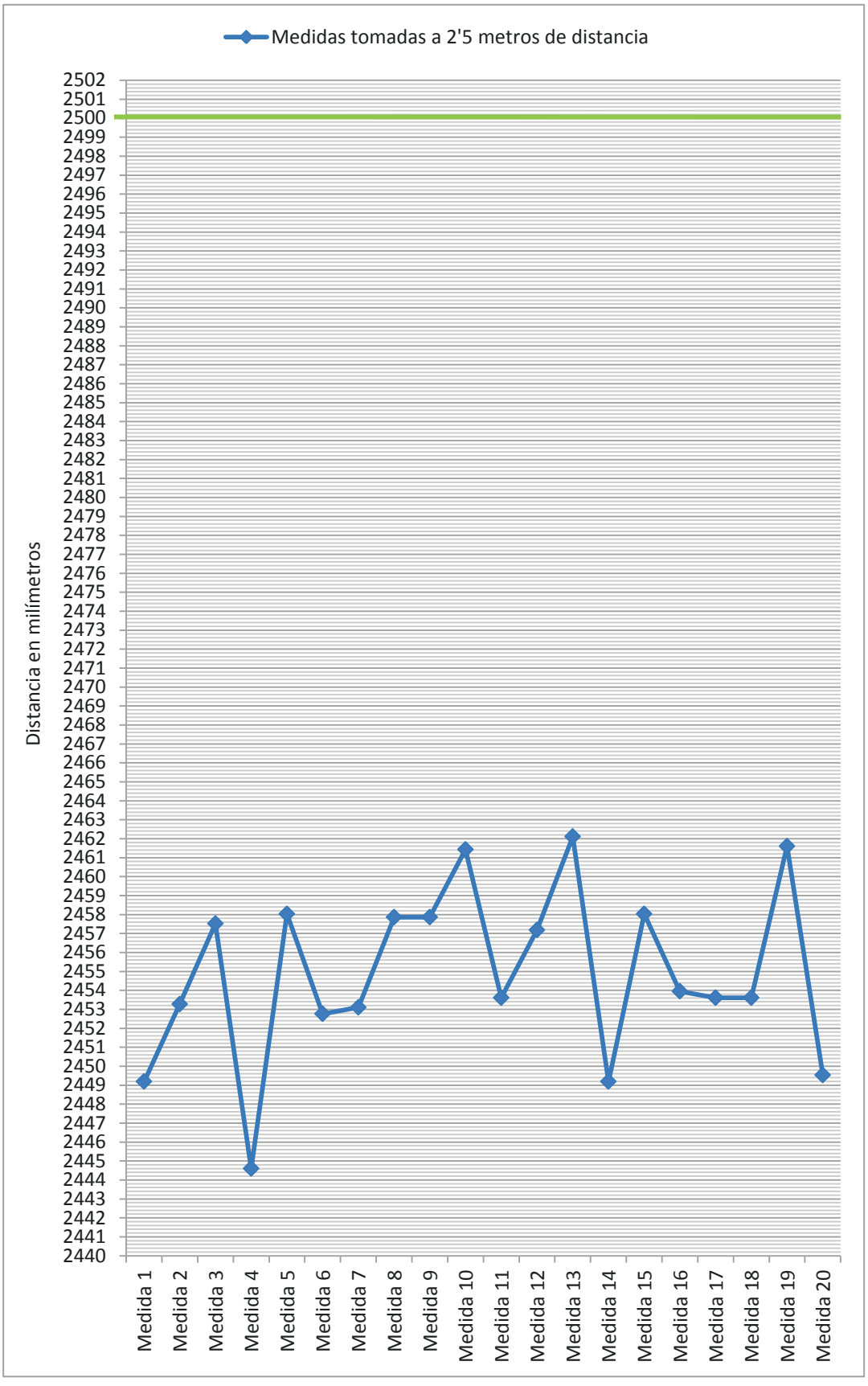
Prueba 1. 14

Prueba 1.15. Desde el sensor hasta una pared a 2[m] de distancia.



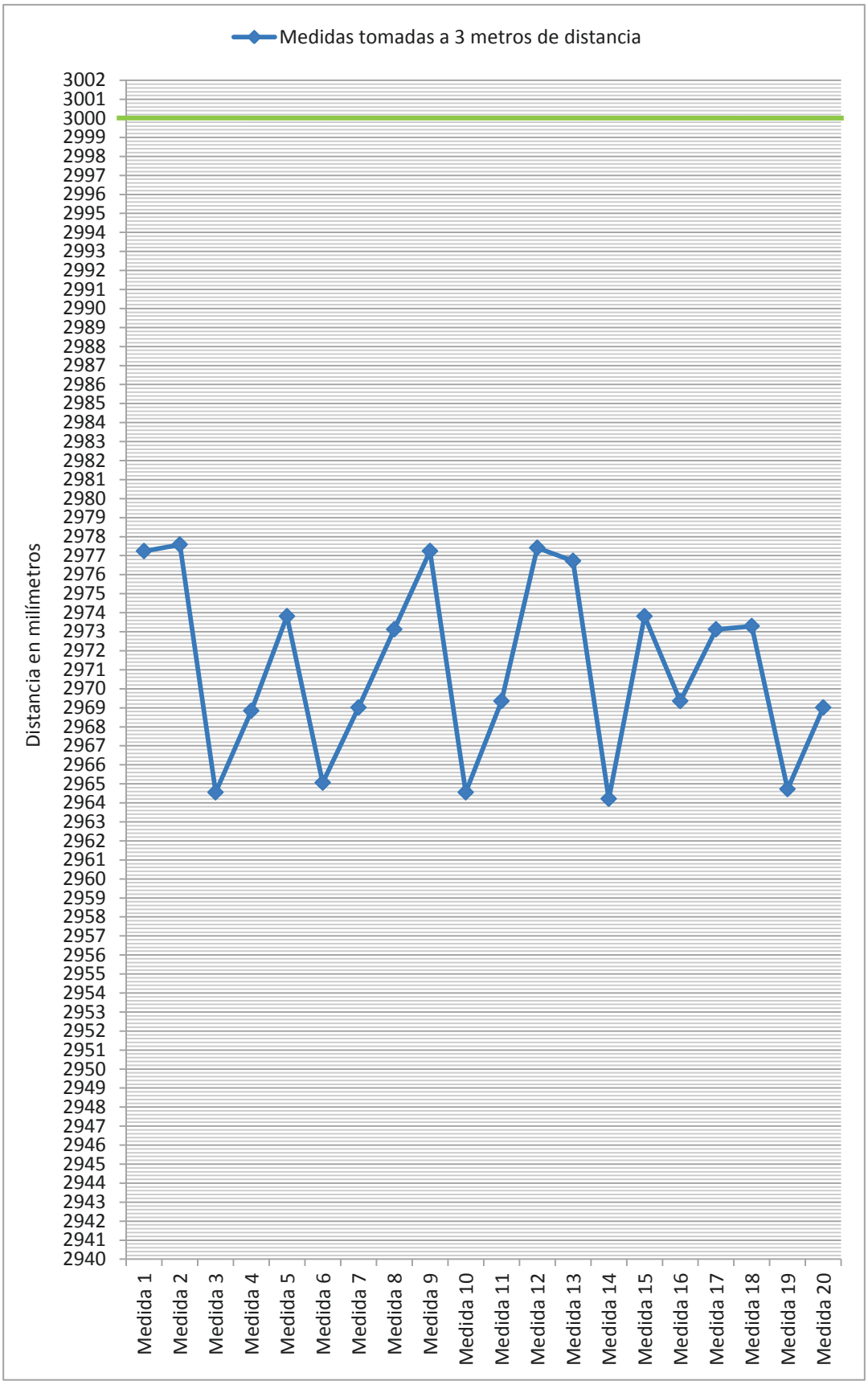
Prueba 1. 15

Prueba 1.16. Desde el sensor hasta una pared a 2'5[m] de distancia.



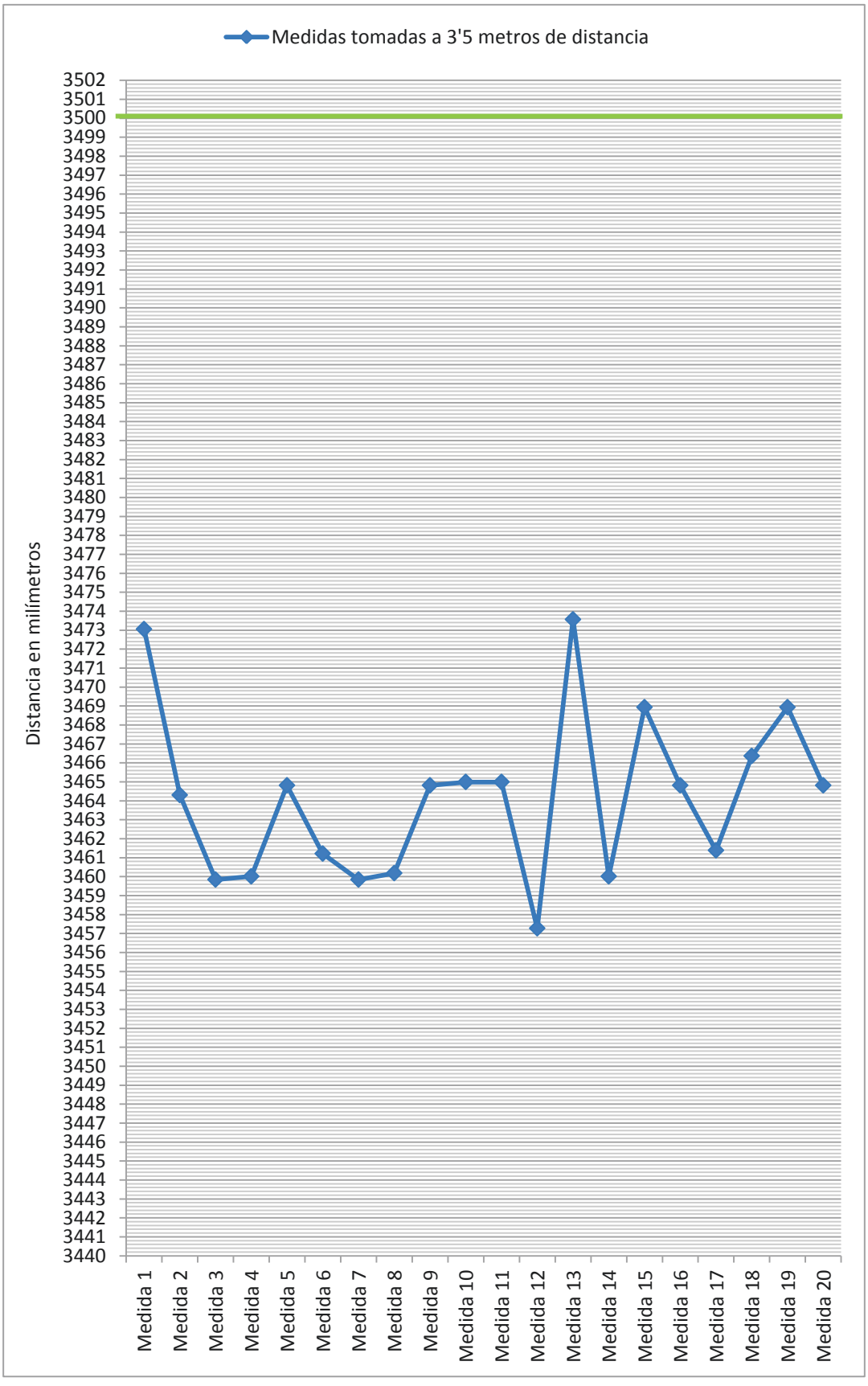
Prueba 1. 16

Prueba 1.17. Desde el sensor hasta una pared a 3[m] de distancia.



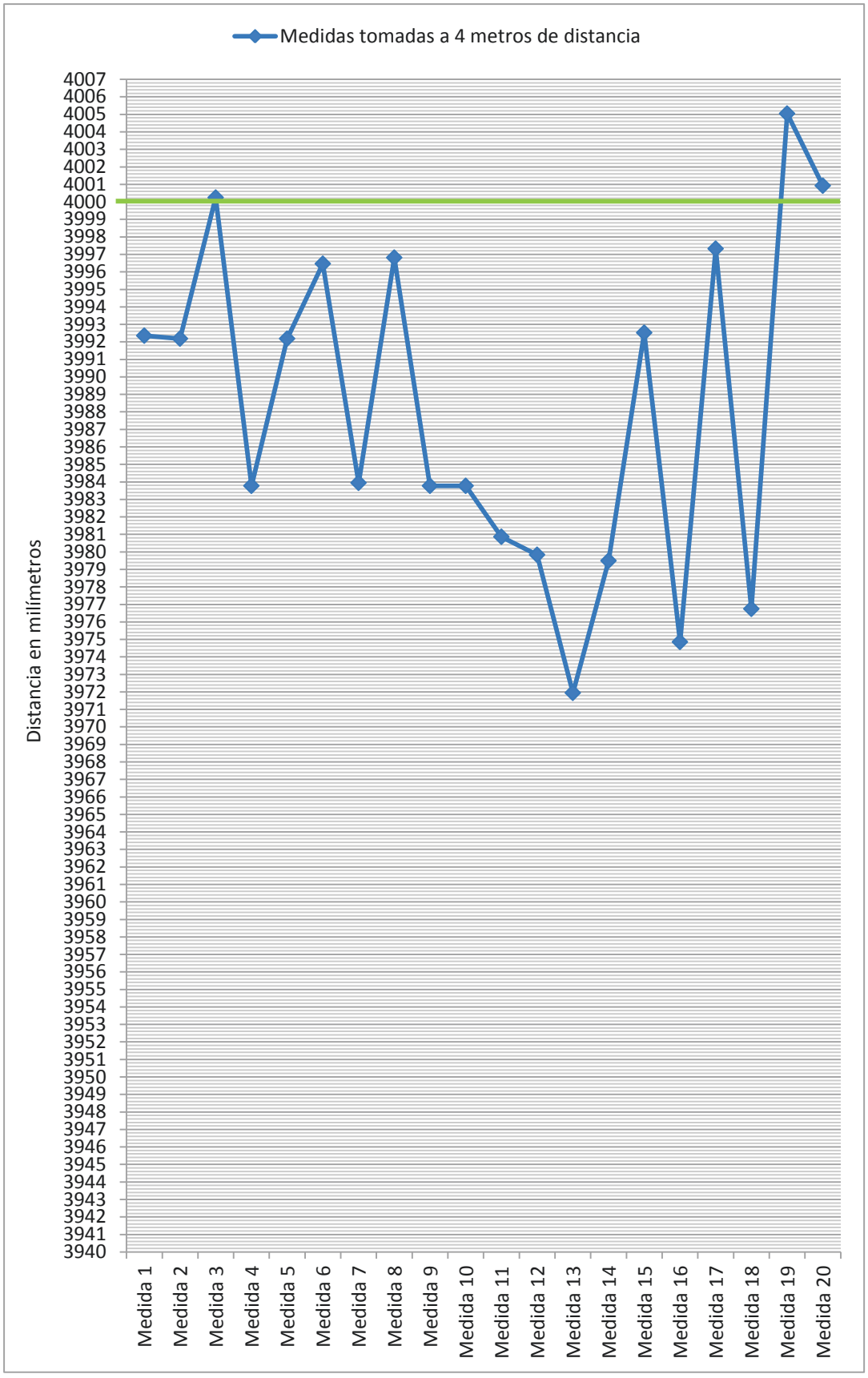
Prueba 1. 17

Prueba 1.18. Desde el sensor hasta una pared a 3'5[m] de distancia.



Prueba 1. 18

Prueba 1.19. Desde el sensor hasta una pared a 4[m] de distancia.

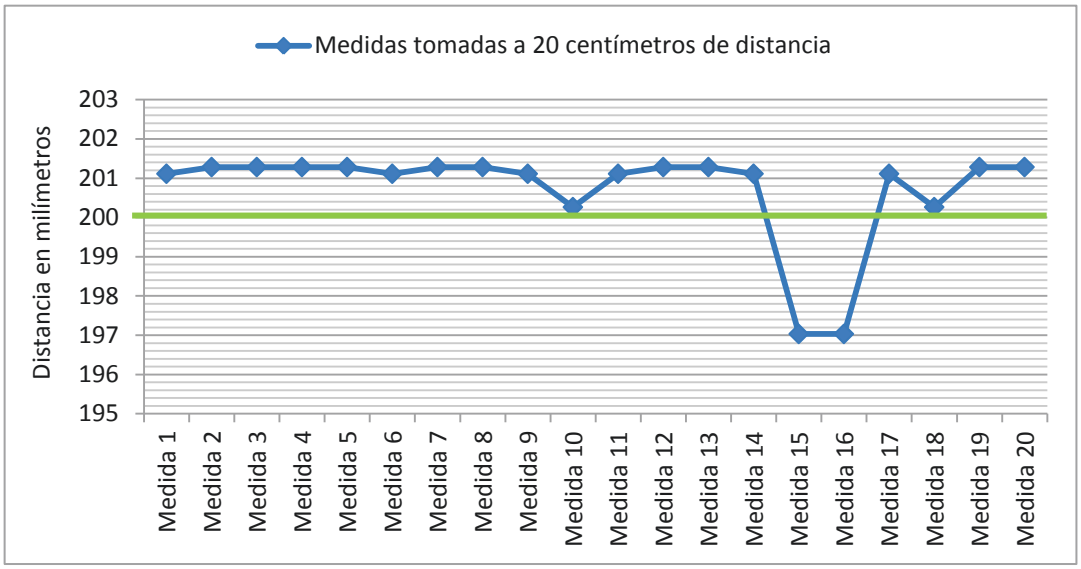


Prueba 1. 19

Prueba 2. Distancia a otra superficie vertical (90° respecto al suelo).

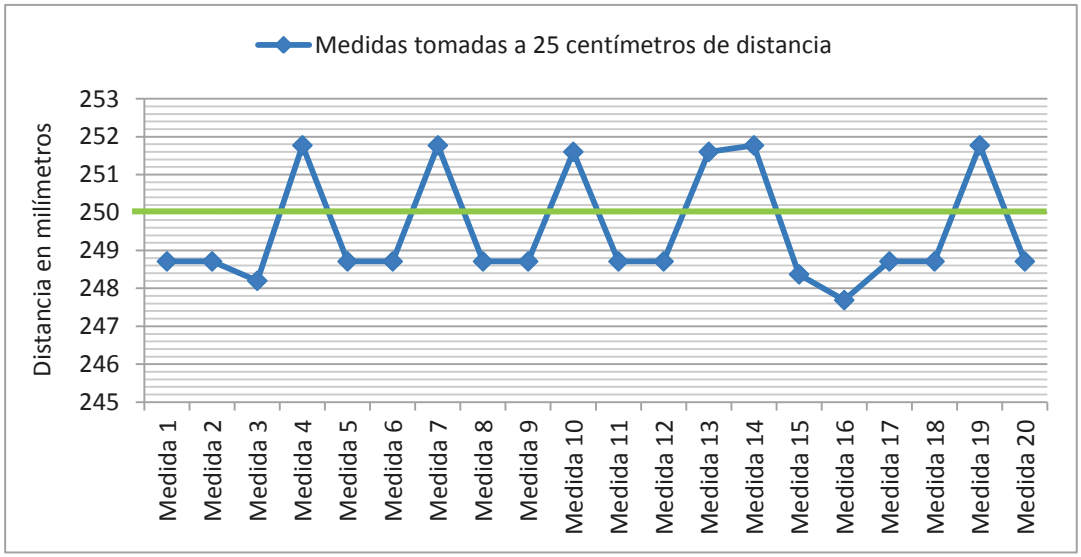
Estas pruebas muestran los resultados de medición obtenidos gracias a Arduino UNO junto al HC-SR04 a partir del *sketch* desarrollado. Las medidas han sido tomadas desde el sensor, hasta una superficie vertical con un área limitada, en concreto, se ha utilizado para estas pruebas la superficie formada por un conjunto de libros apilados de forma horizontal, unos sobre otros, constituyendo un obstáculo de 27[cm] de largo × 8[cm] de alto, en el cual reboten las ondas de ultrasonidos.

Prueba 2.1. Desde el sensor hasta una superficie concreta, a 20[cm] de distancia.



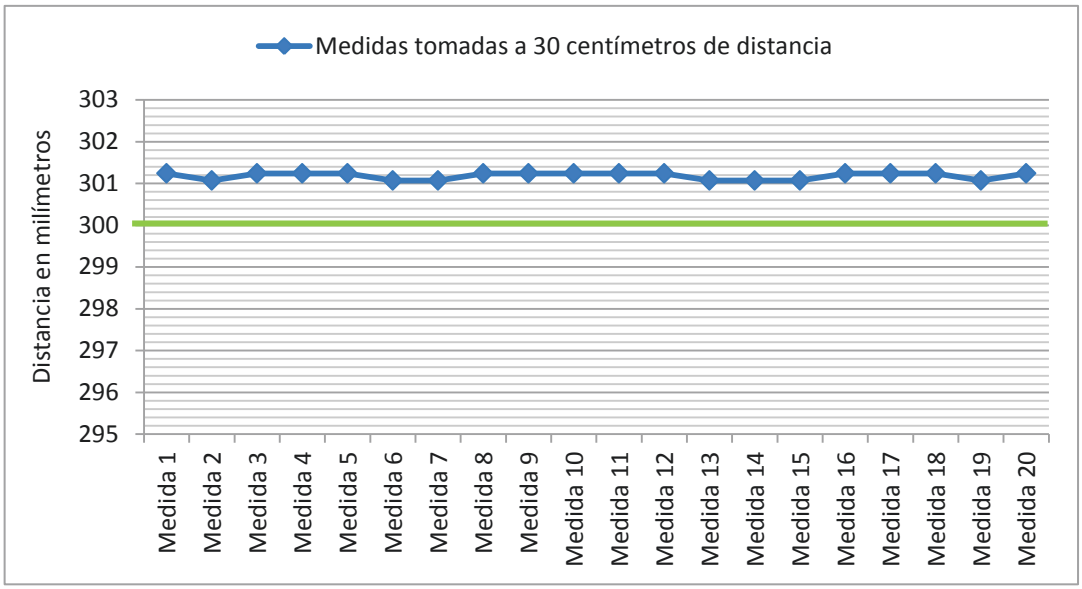
Prueba 2. 1

Prueba 2.2. Desde el sensor hasta una superficie concreta, a 25[cm] de distancia.



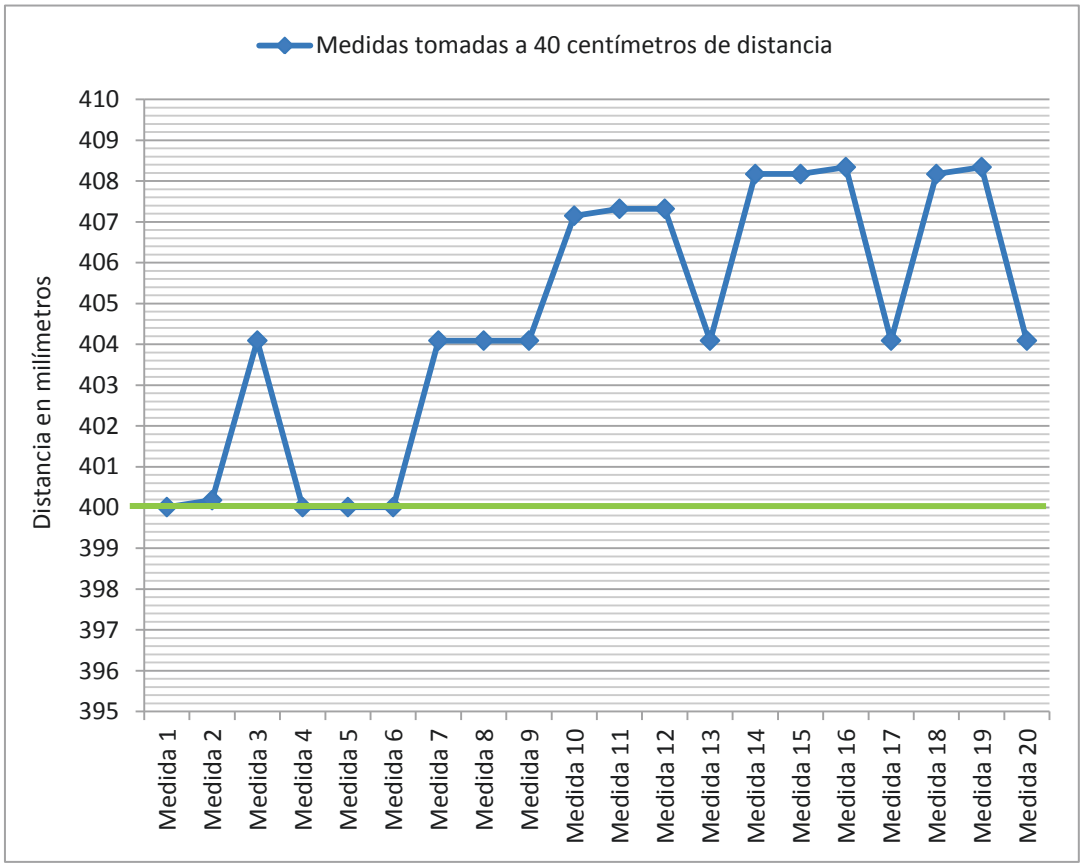
Prueba 2. 2

Prueba 2.3. Desde el sensor hasta una superficie concreta, a 30[cm] de distancia.



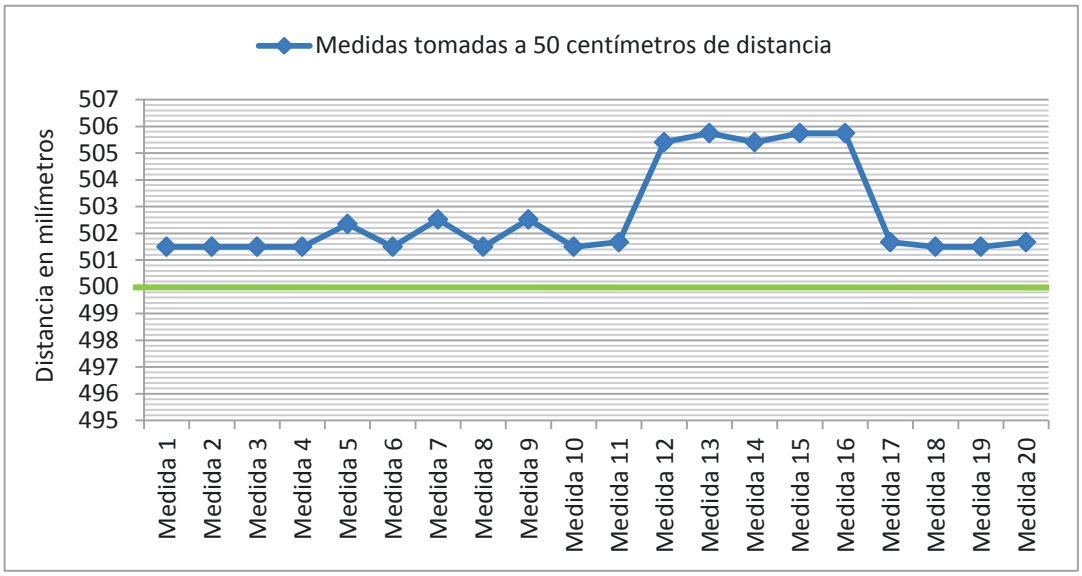
Prueba 2. 3

Prueba 2.4. Desde el sensor hasta una superficie concreta, a 40[cm] de distancia.



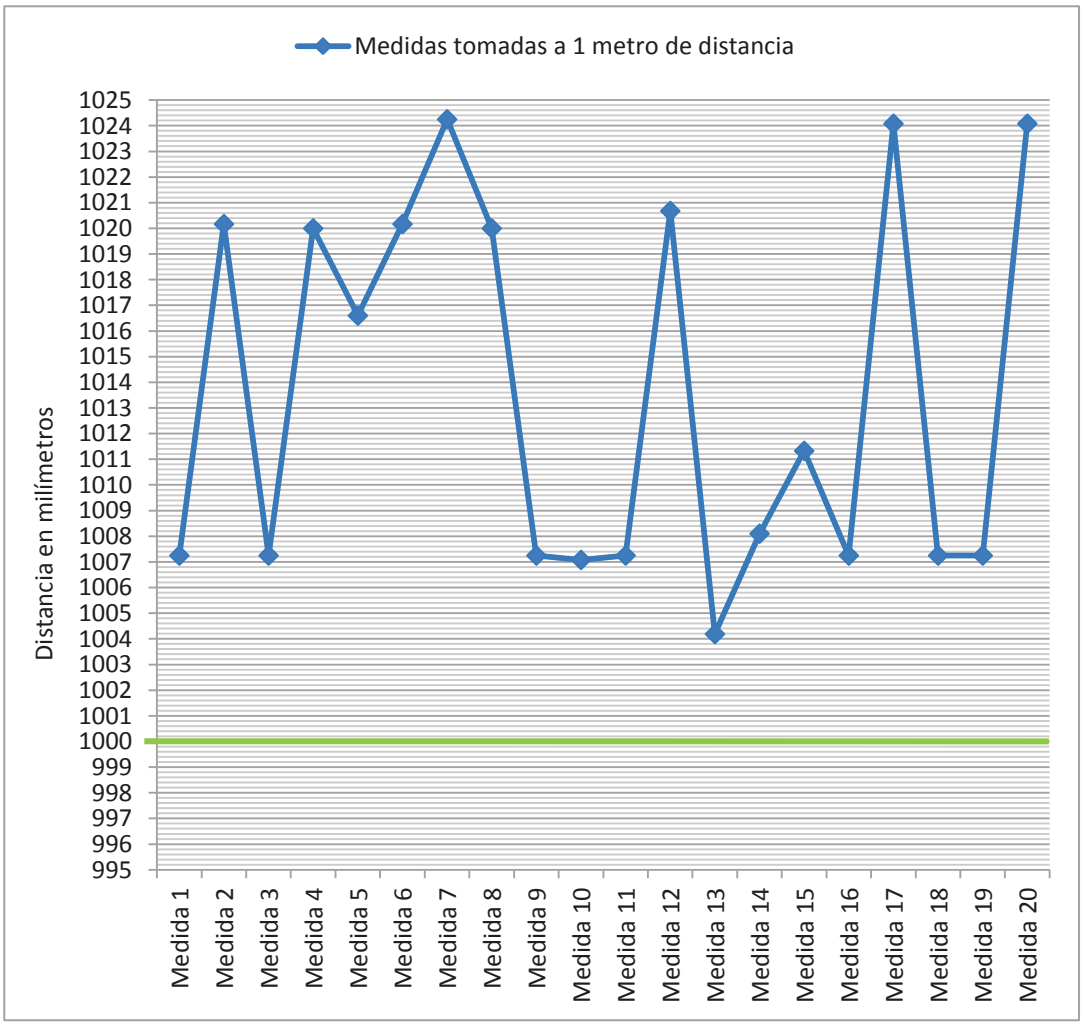
Prueba 2. 4

Prueba 2.5. Desde el sensor hasta una superficie concreta, a 50[cm] de distancia.



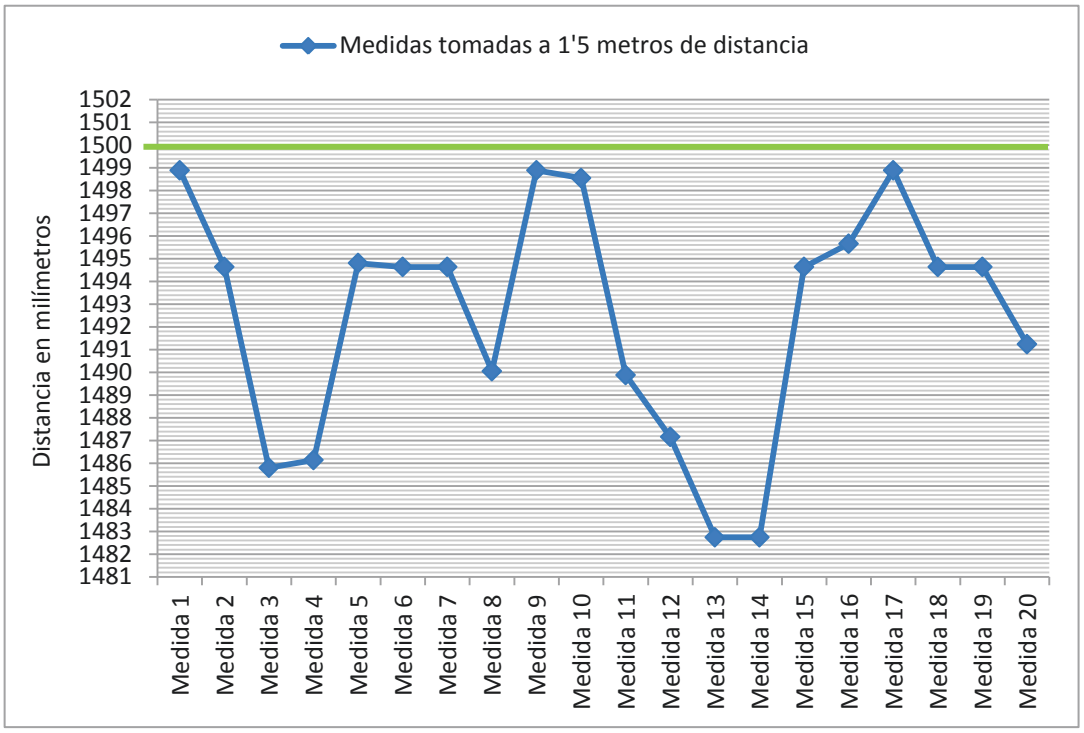
Prueba 2. 5

Prueba 2.6. Desde el sensor hasta una superficie concreta, a 1[m] de distancia.



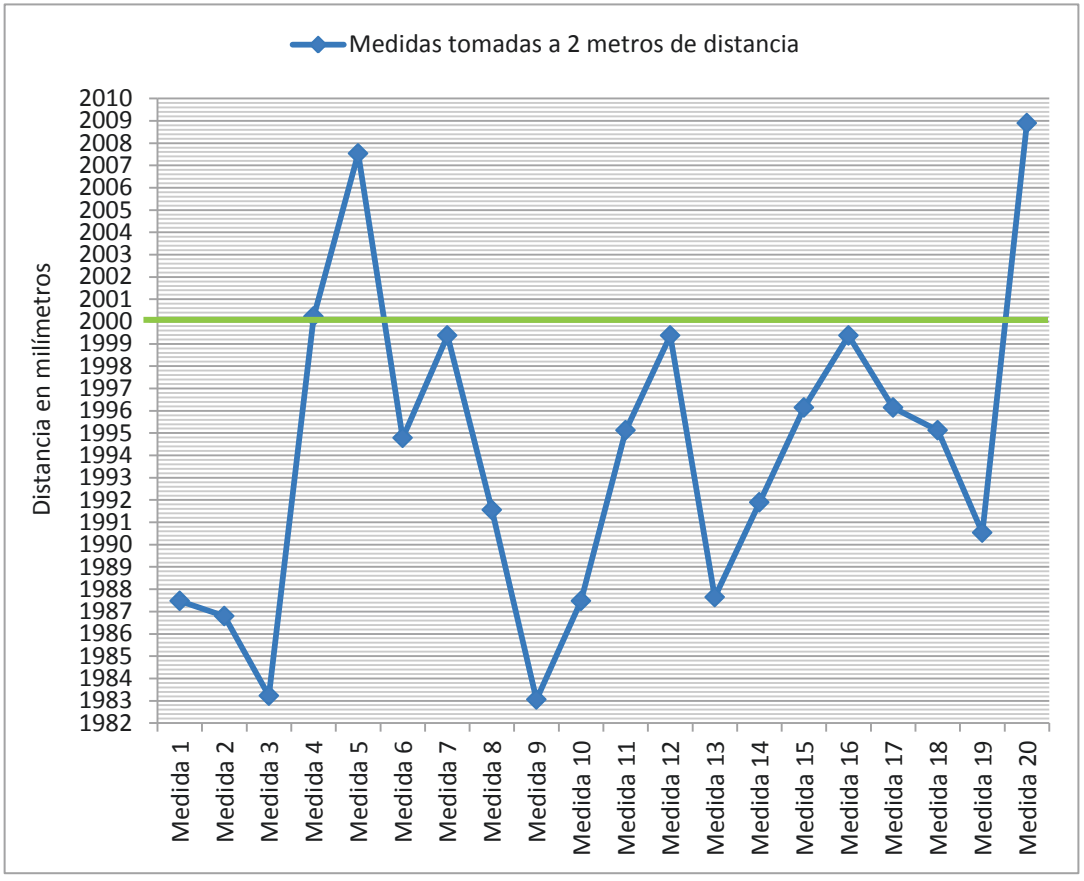
Prueba 2. 6

Prueba 2.7. Desde el sensor hasta una superficie concreta, a 1'5[m] de distancia.



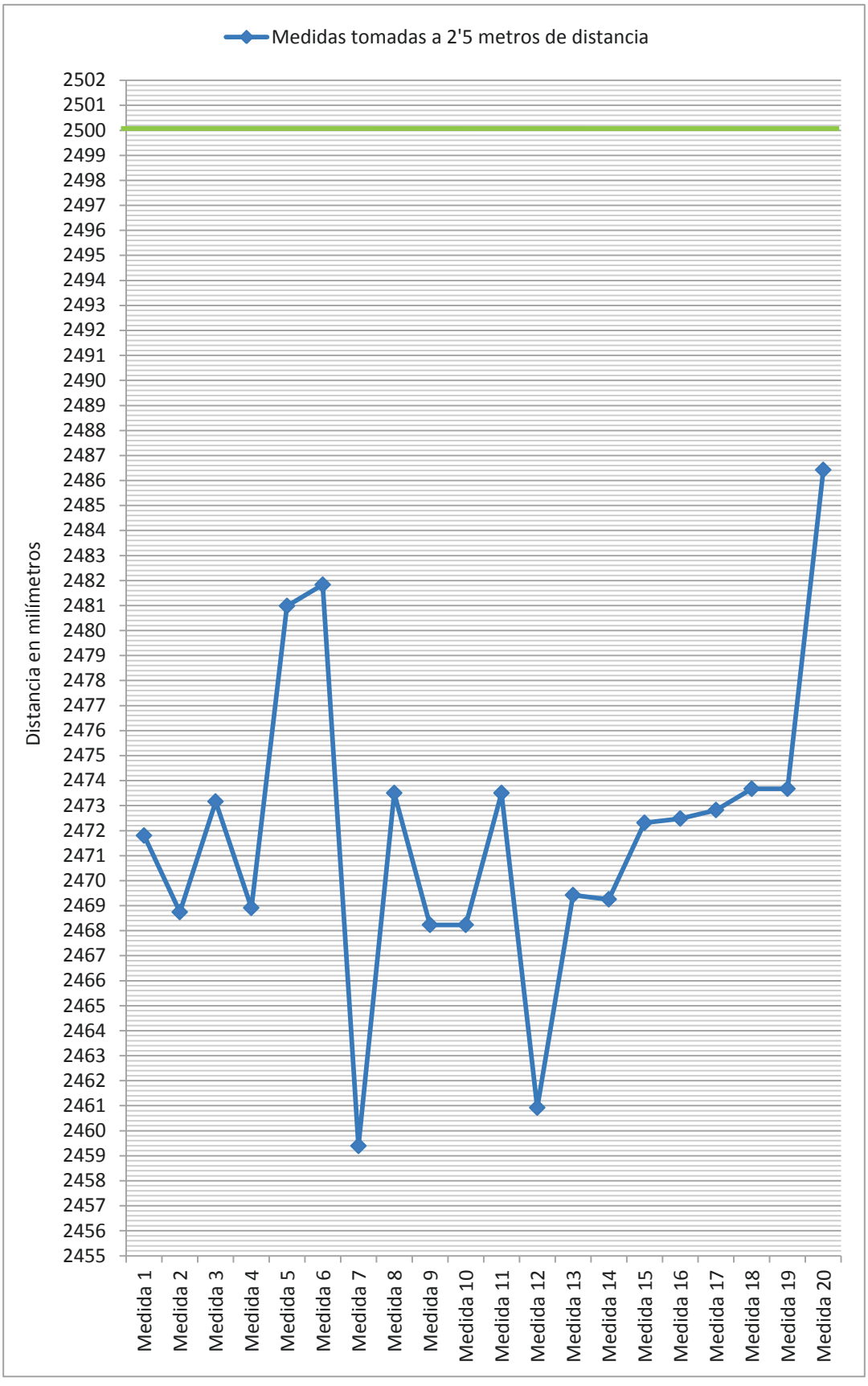
Prueba 2. 7

Prueba 2.8. Desde el sensor hasta una superficie concreta, a 2[m] de distancia.



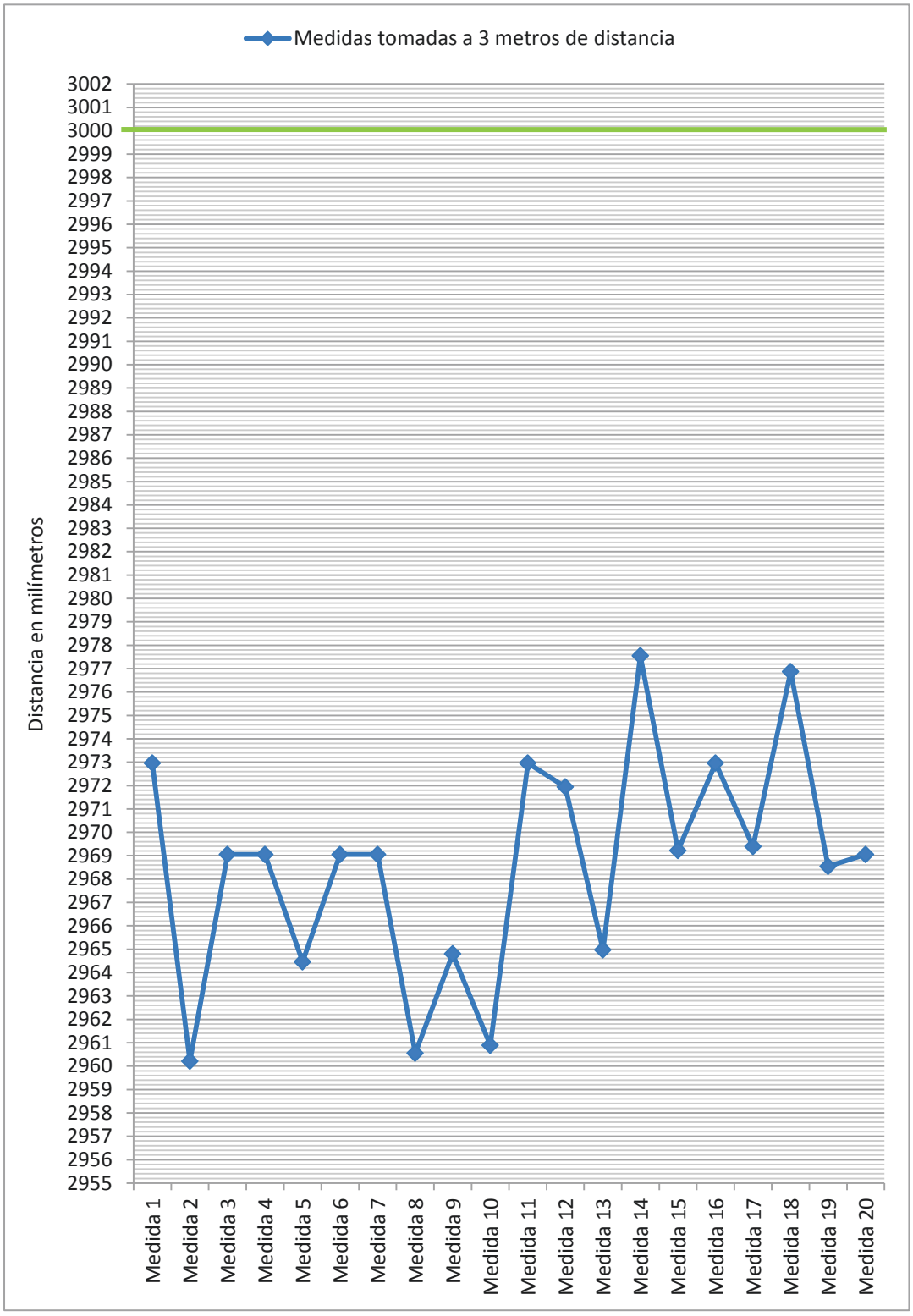
Prueba 2. 8

Prueba 2.9. Desde el sensor hasta una superficie concreta, a 2'5[m] de distancia.



Prueba 2. 9

Prueba 2.10. Desde el sensor hasta una superficie concreta, a 3[m] de distancia.



Prueba 2. 10

Prueba 2.11. Desde el sensor hasta una superficie concreta, a 3'5[m] de distancia.

Al tomar las medidas de distancia hasta una superficie como la que se ha empleado en realizar esta prueba (formada un conjunto de libros apilados de forma horizontal, unos encima de otros, constituyendo una superficie de 27[cm] de largo × 8[cm] de alto), y en una habitación con mobiliario, las ondas de ultrasonidos (cuya amplitud de ángulo se ve incrementado proporcionalmente con el aumento de la distancia) dejan de rebotar únicamente en la superficie que se desea medir, para hacerlo sobre otras superficies cercanas, desviándose continuamente, y provocando que, resultado de la medición, se obtengan muchos picos de error y la precisión del sensor deje de ser fiable y precisa, alejándose considerablemente de la distancia que existe realmente hasta el objeto.

En la tabla que aquí aparece se muestran los resultados obtenidos en este experimento.

	Medidas tomadas a 3'5 [m]
Medida 1	-3499,79
Medida 2	1944,12
Medida 3	51,34
Medida 4	4471,51
Medida 5	51,34
Medida 6	-3424,65
Medida 7	51,34
Medida 8	1936,13
Medida 9	4427,48
Medida 10	4423,57
Medida 11	1945,48
Medida 12	51,34
Medida 13	51,34
Medida 14	50,49
Medida 15	4424,59
Medida 16	1953,13
Medida 17	1961,29
Medida 18	4428,84
Medida 19	4477,29
Medida 20	50,49

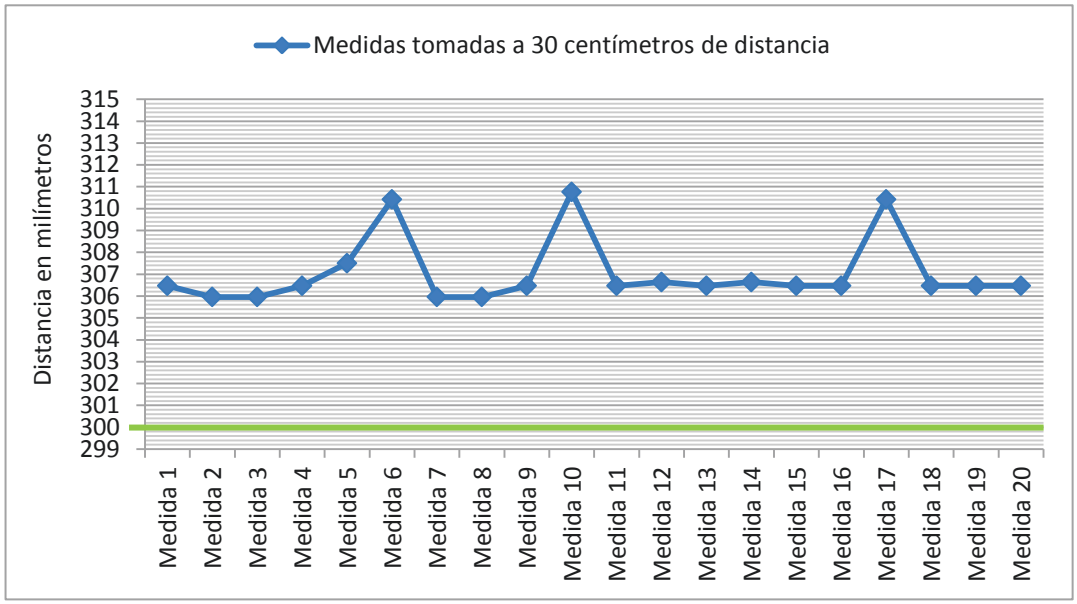
Prueba 2. 101

Teóricamente, en un espacio en el no hubiese otros objetos dentro de un ángulo de 15° desde el sensor hasta el objeto a medir (para evitar que las ondas ultrasónicas se desviasen), el sensor tendría una capacidad de medición fiable aproximadamente hasta 4[m] de distancia.

Prueba 3. Distancia a una superficie inclinada respecto al suelo.

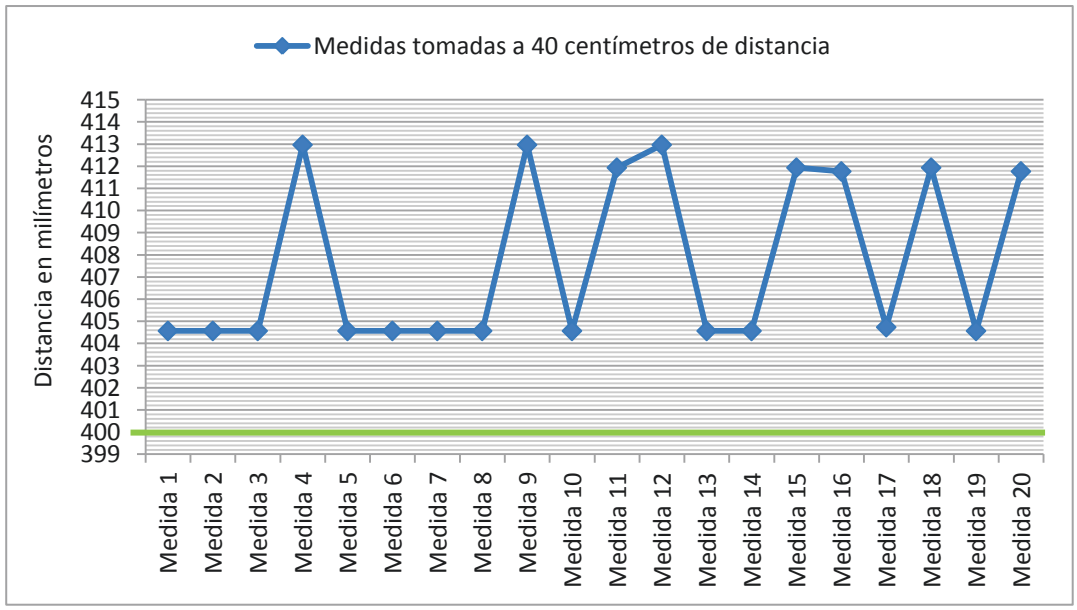
Las siguientes pruebas muestran el estudio de los resultados de medición obtenidos gracias a Arduino UNO junto al HC-SR04 a partir del *sketch* desarrollado. Las medidas han sido tomadas desde el sensor, hasta una superficie vertical con un área limitada e inclinada respecto al suelo, tal como un marco de fotos, inclinado ≃20° respecto al suelo.

Prueba 3.1. Desde el sensor hasta un marco de fotos a 30[cm] de distancia.



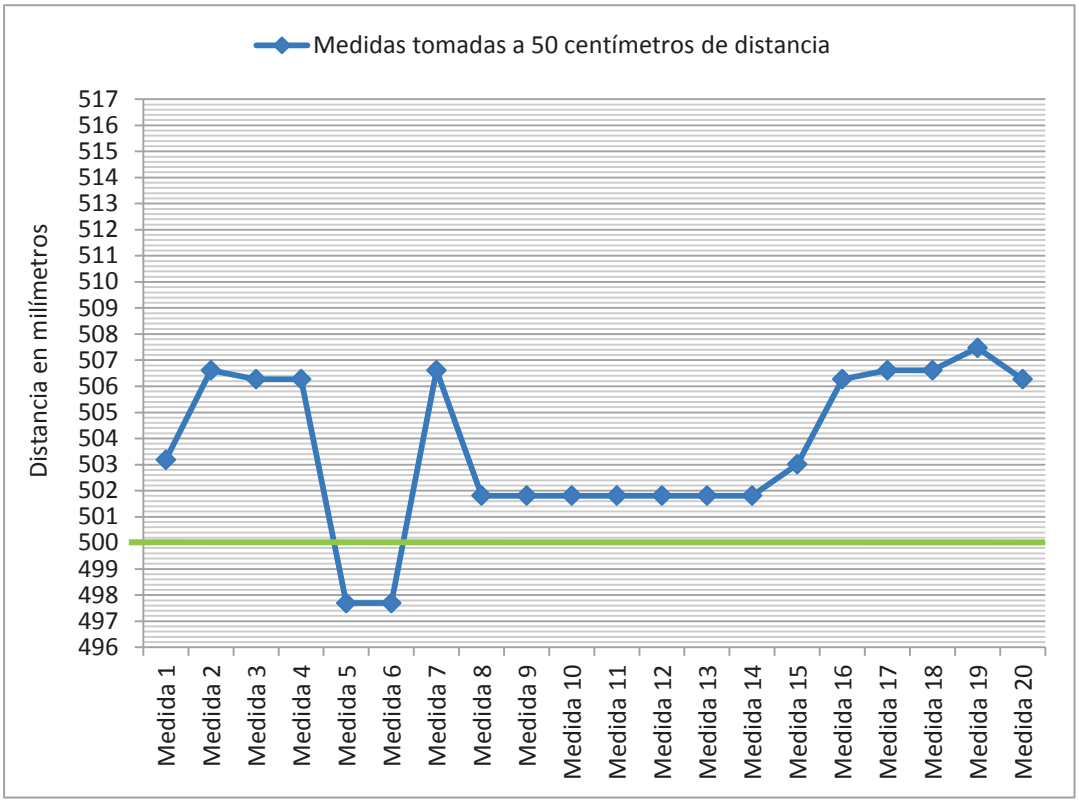
Prueba 3. 1

Prueba 3.2. Desde el sensor hasta un marco de fotos a 40[cm] de distancia.



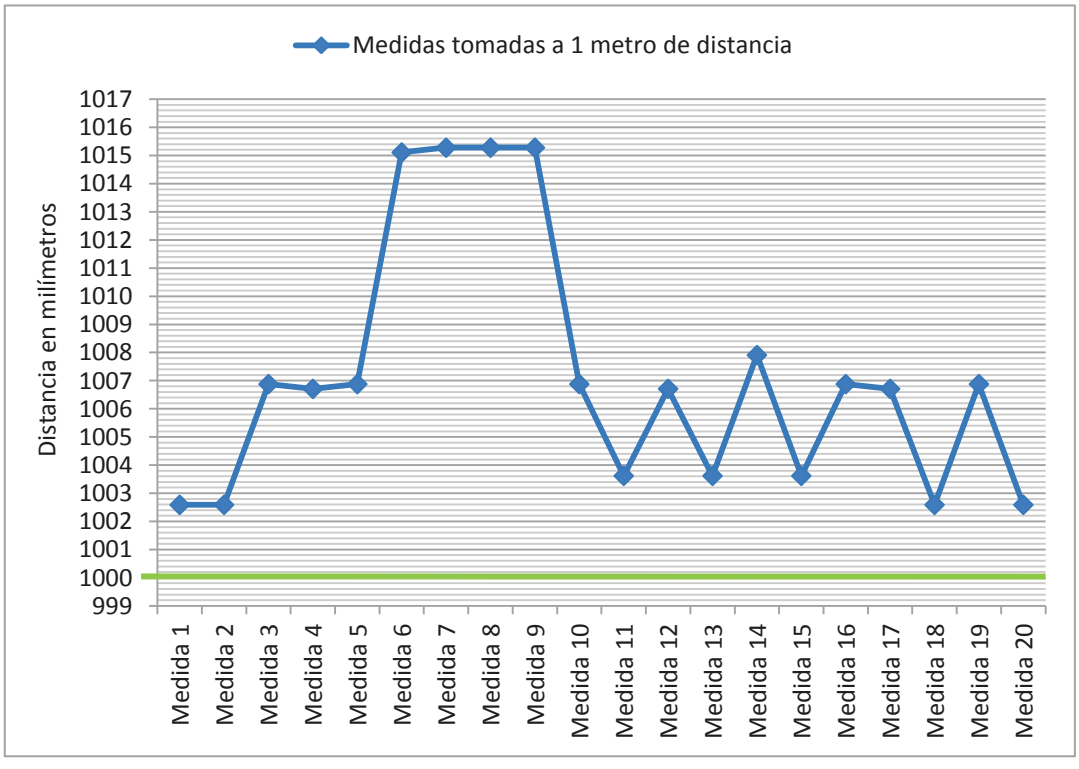
Prueba 3. 2

Prueba 3.3. Desde el sensor hasta un marco de fotos a 50[cm] de distancia.



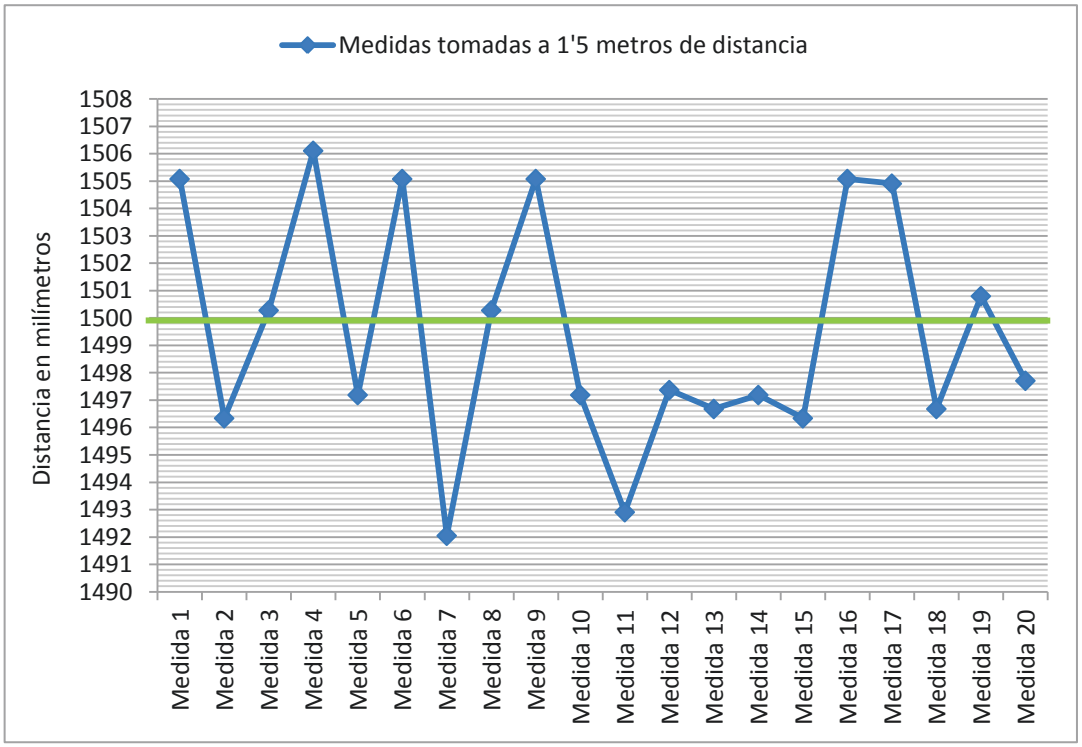
Prueba 3. 3

Prueba 3.4. Desde el sensor hasta un marco de fotos a 1[m] de distancia.



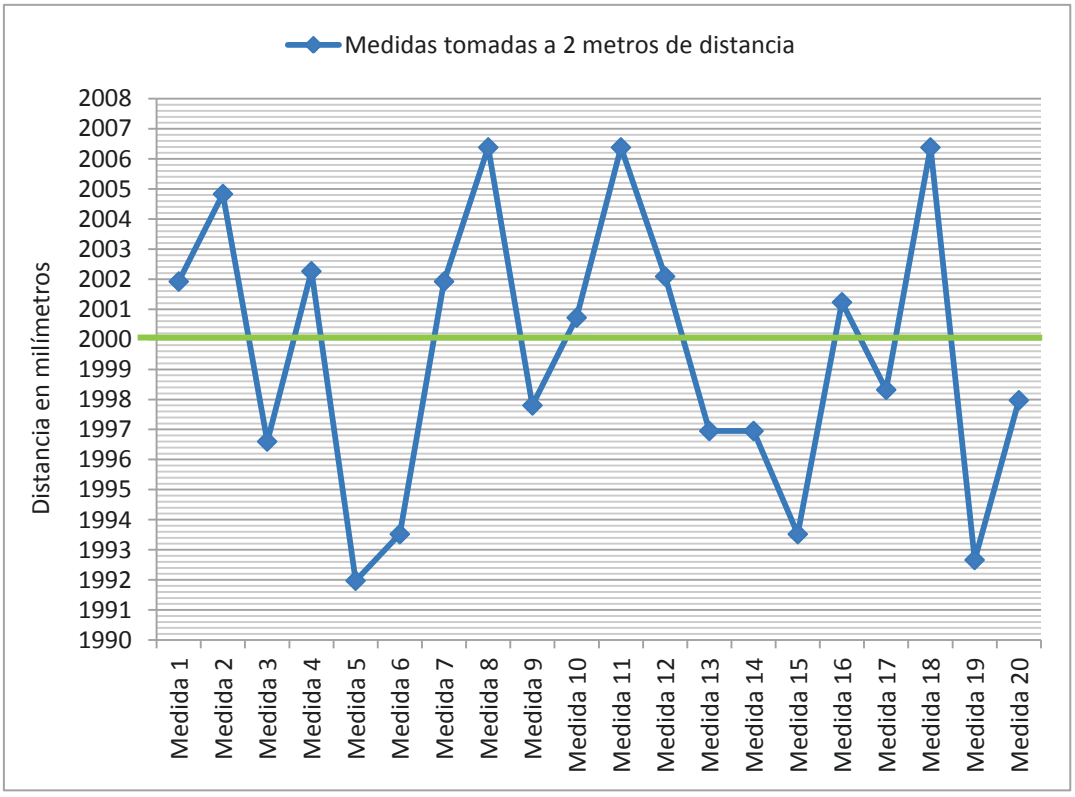
Prueba 3. 4

Prueba 3.5. Desde el sensor hasta un marco de fotos a 1'5[m] de distancia.



Prueba 3. 5

Prueba 3.6. Desde el sensor hasta un marco de fotos a 2[m] de distancia.



Prueba 3. 6

Prueba 3.7. Desde el sensor hasta un marco de fotos a 2'5[m] de distancia.

De la misma forma que ocurría en la medición de distancias a una superficie limitada en área, al tomar las medidas de distancia hasta una superficie como la que se ha empleado en realizar esta prueba (un marco de fotografías), y en una habitación con mobiliario, las ondas de ultrasonidos dejan de rebotar únicamente en la superficie que se desea medir, para hacerlo sobre otras superficies cercanas, desviándose y provocando que, resultado de la medición, se empiecen a obtener errores de medición.

En la tabla que aquí aparece se muestran los resultados obtenidos en este experimento.

Teóricamente, en un espacio en el no hubiese otros objetos dentro de un ángulo de 15° desde el sensor hasta el objeto a medir (para evitar que las ondas ultrasónicas se desviasen), el sensor tendría una capacidad de medición fiable aproximadamente hasta 4[m] de distancia.

	Medidas tomadas a 2'5 [m]
Medida 1	2504,93
Medida 2	51,79
Medida 3	2506,13
Medida 4	2500,64
Medida 5	3870,58
Medida 6	2506,13
Medida 7	3862,69
Medida 8	2500,47
Medida 9	3884,13
Medida 10	3870,58
Medida 11	3883,45
Medida 12	3863,21
Medida 13	2514,88
Medida 14	2515,05
Medida 15	2506,13
Medida 16	2502,18
Medida 17	3879,84
Medida 18	2497,9
Medida 19	2502,01
Medida 20	2492,07

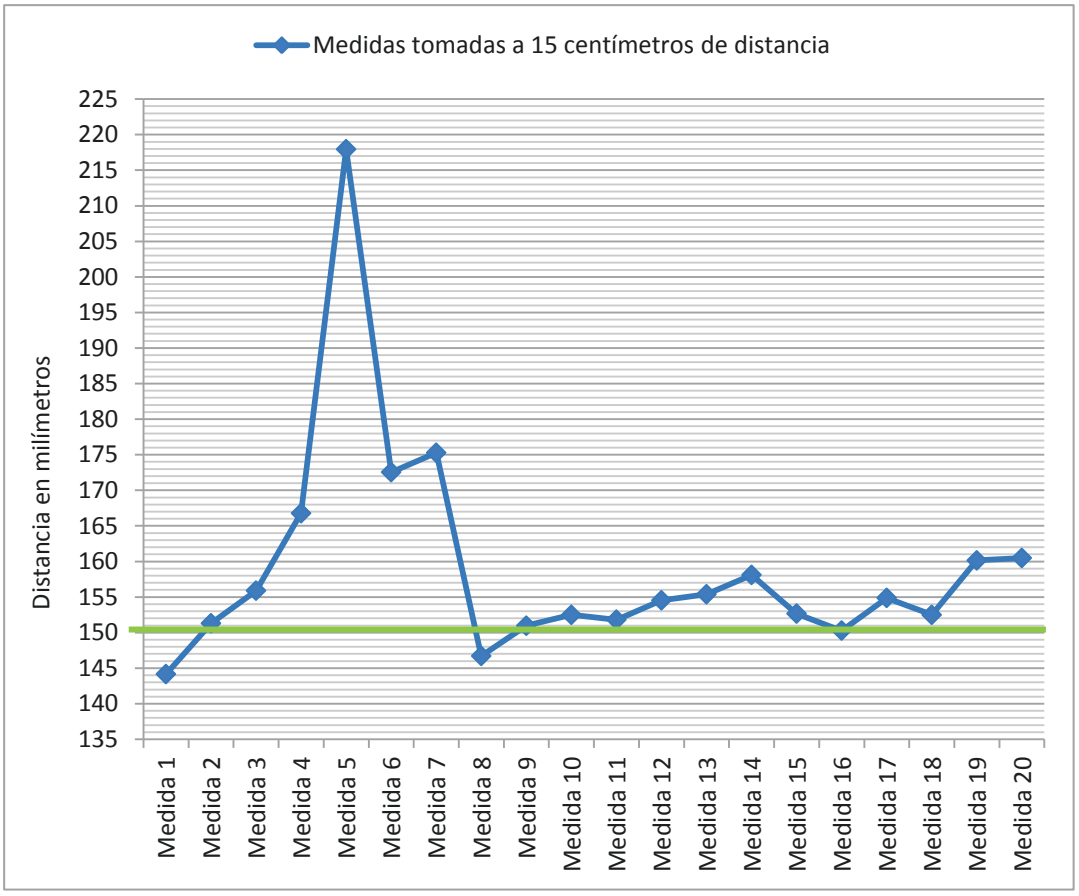
Prueba 3. 7

Prueba 4. Distancia a una superficie vertical cilíndrica translúcida.

Las siguientes pruebas muestran los resultados de medición obtenidos gracias al conjunto formado por la placa Arduino UNO junto al sensor HC-SR04, a partir del *sketch* implementado. Las medidas han sido tomadas desde el sensor, hasta una superficie vertical cilíndrica (de 11[cm] de radio) rugosa y translúcida, como puede ser la pantalla de una lámpara de pie.

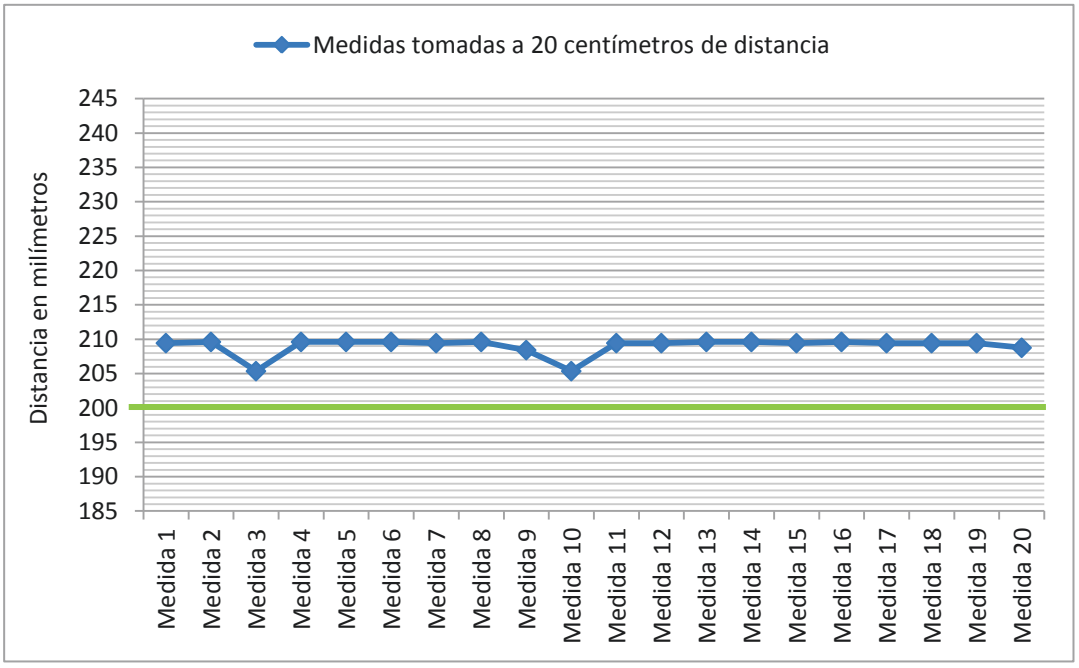
La medida tomada como valor de distancia real, será en cualquier caso aproximada, ya que, al tratarse de una superficie cilíndrica, es poco probable apuntar exactamente al mismo punto con el sensor de ultrasonidos y con el instrumento de medida empleado como referencia.

Prueba 4.1. Desde el sensor hasta una lámpara cilíndrica, a 15[cm] de distancia.



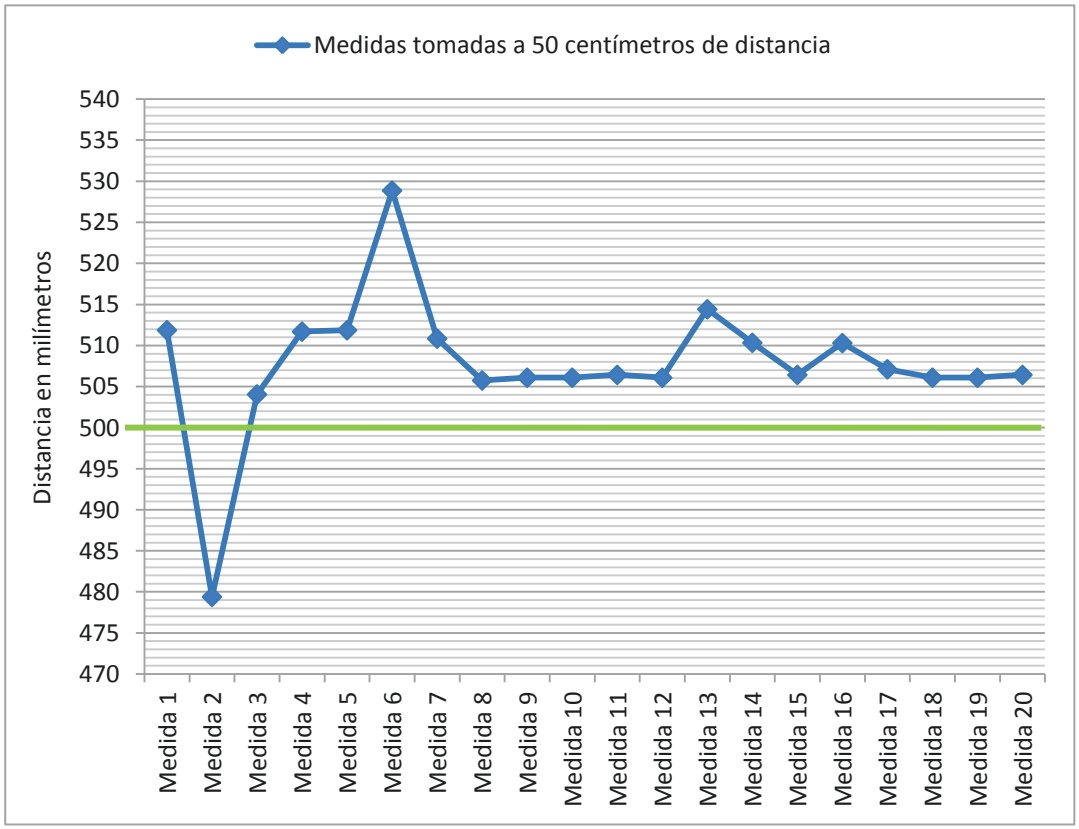
Prueba 4. 1

Prueba 4.2. Desde el sensor hasta una lámpara cilíndrica, a 20[cm] de distancia.



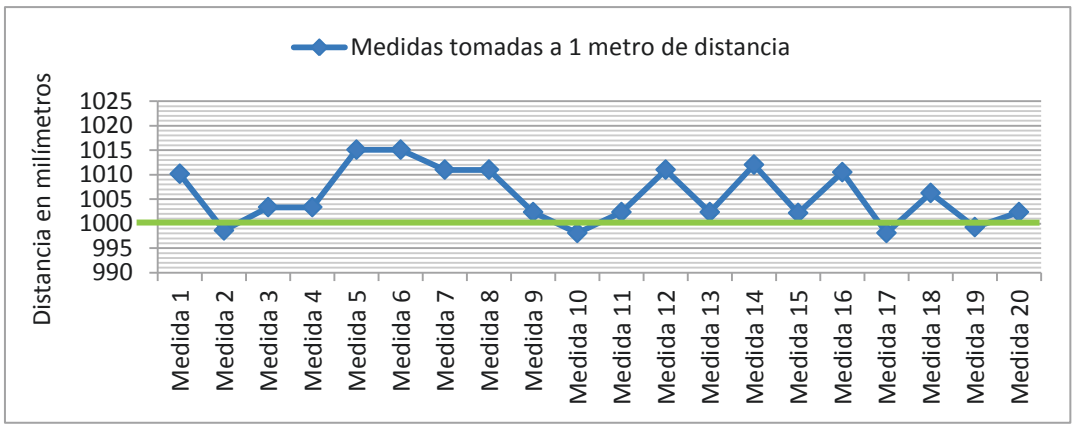
Prueba 4. 2

Prueba 4.3. Desde el sensor hasta una lámpara cilíndrica, a 50[cm] de distancia.



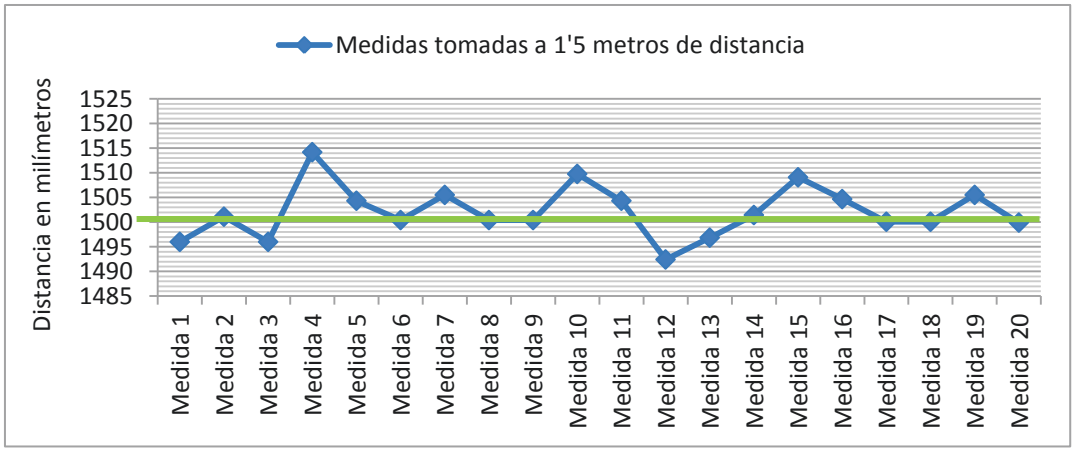
Prueba 4. 3

Prueba 4.4. Desde el sensor hasta una lámpara cilíndrica, a 1[m] de distancia.



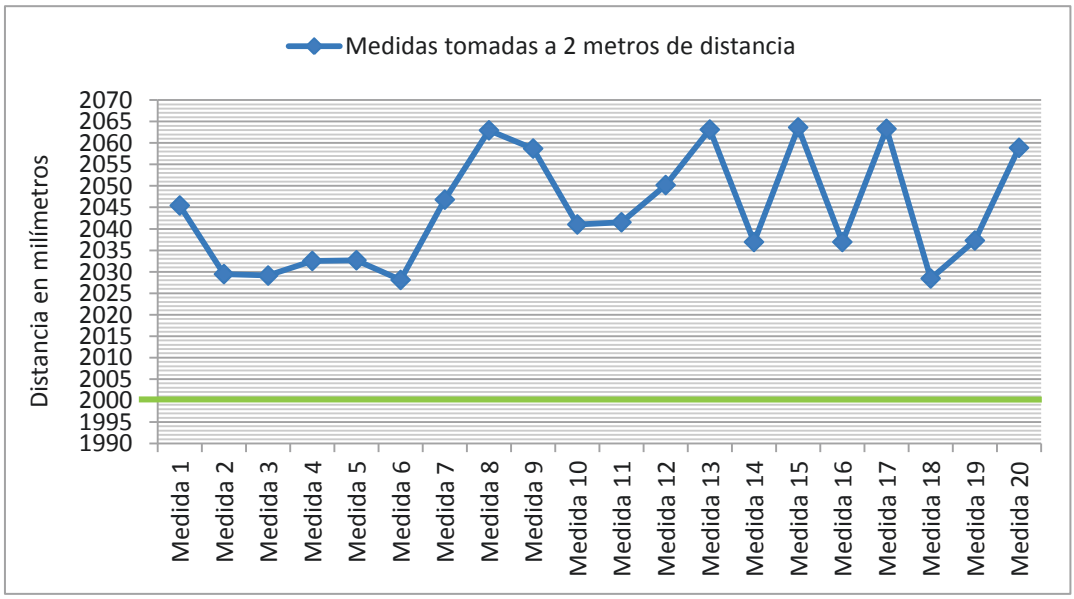
Prueba 4. 4

Prueba 4.5. Desde el sensor hasta una lámpara cilíndrica, a 1'5[m] de distancia.



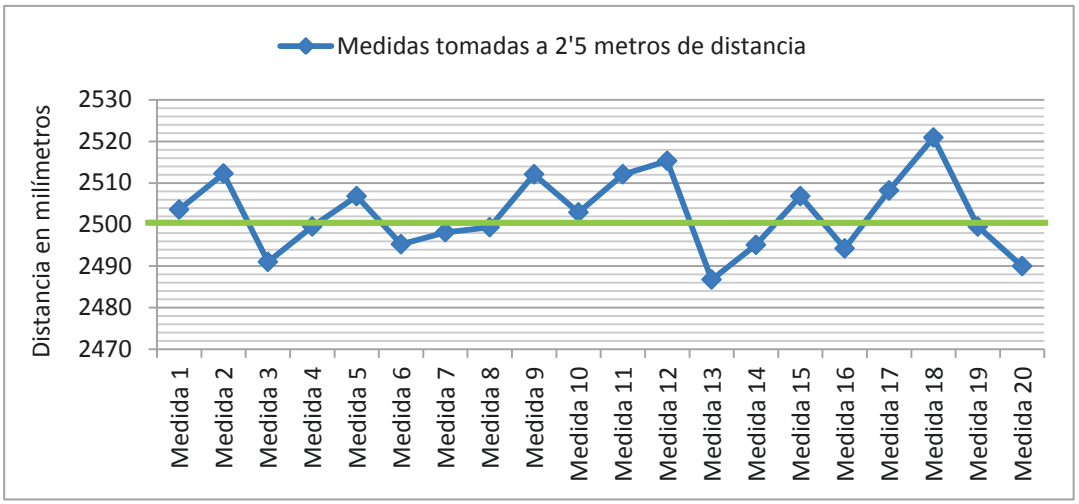
Prueba 4. 5

Prueba 4.6. Desde el sensor hasta una lámpara cilíndrica, a 2[m] de distancia.



Prueba 4. 6

Prueba 4.7. Desde el sensor hasta una lámpara cilíndrica, a 2'5[m] de distancia.



Prueba 4. 7

Prueba 4.8. Desde el sensor hasta una lámpara cilíndrica, a 3[m] de distancia.

A esta distancia, las medidas de distancia tomadas por el sensor, no se ajustan a la distancia real. De la misma forma que ocurría en los casos anteriores en los que se ha experimentado con la medición de distancias a una superficie limitada en área, al tomar las medidas de distancia hasta superficie cilíndrica, las ondas de ultrasonidos dejan de rebotar únicamente dicha superficie y se desvían provocando mediciones que distan de la distancia real del sensor a la lámpara.

En la tabla se muestran los resultados derivados de este experimento.

Teóricamente, en un espacio en el no hubiese otros objetos dentro de un ángulo de 15° desde el sensor hasta el objeto a medir (para evitar que las ondas ultrasónicas se desviasen), el sensor tendría una capacidad de medición fiable aproximadamente hasta 4[m] de distancia. Lo cual es difícil de conseguir con una superficie cilíndrica.

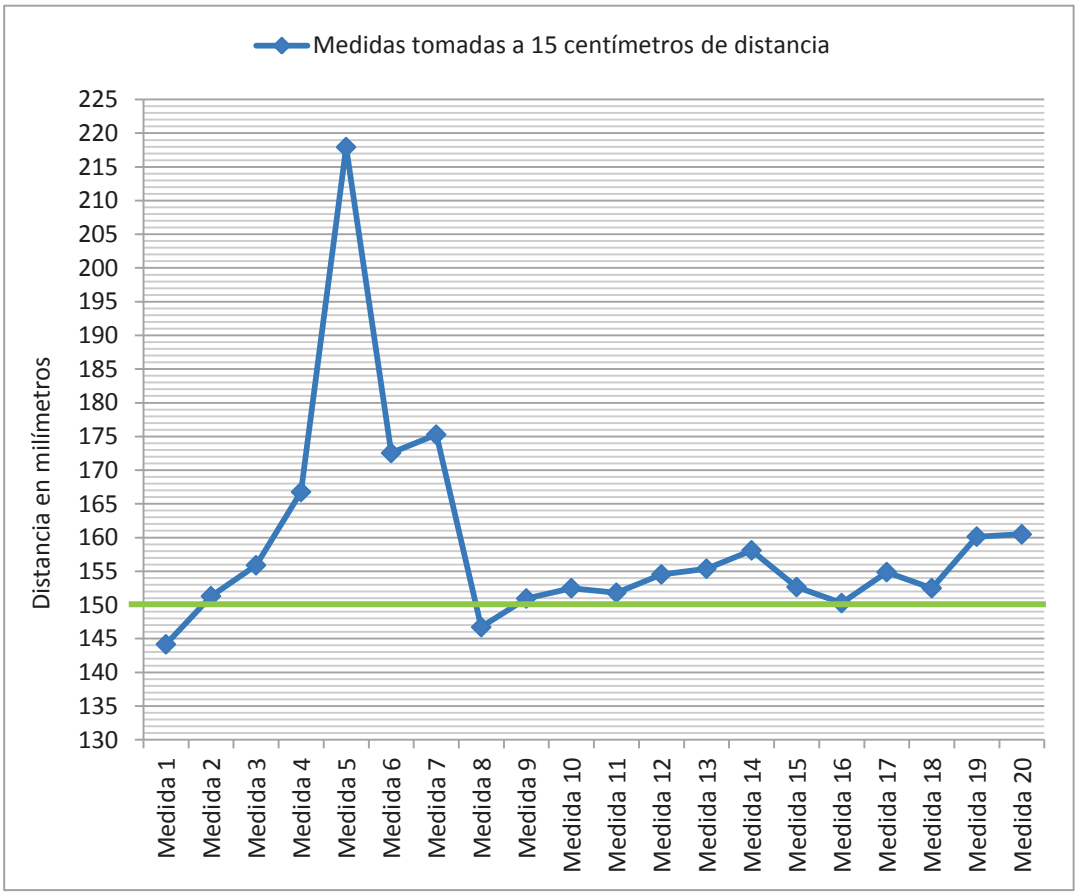
	Medidas tomadas a 3[m]
Medida 1	51,34
Medida 2	71,57
Medida 3	55,59
Medida 4	4108,05
Medida 5	4106,01
Medida 6	4093,26
Medida 7	4098,87
Medida 8	4106,01
Medida 9	2427,94
Medida 10	4144,43
Medida 11	4148,51
Medida 12	4144,43
Medida 13	71,57
Medida 14	4140,01
Medida 15	4144,26
Medida 16	2415,02
Medida 17	2428,11
Medida 18	4140,69
Medida 19	2436,44
Medida 20	4140,69

Prueba 4. 8

Prueba 5. Distancia a la cara de una persona de pie.

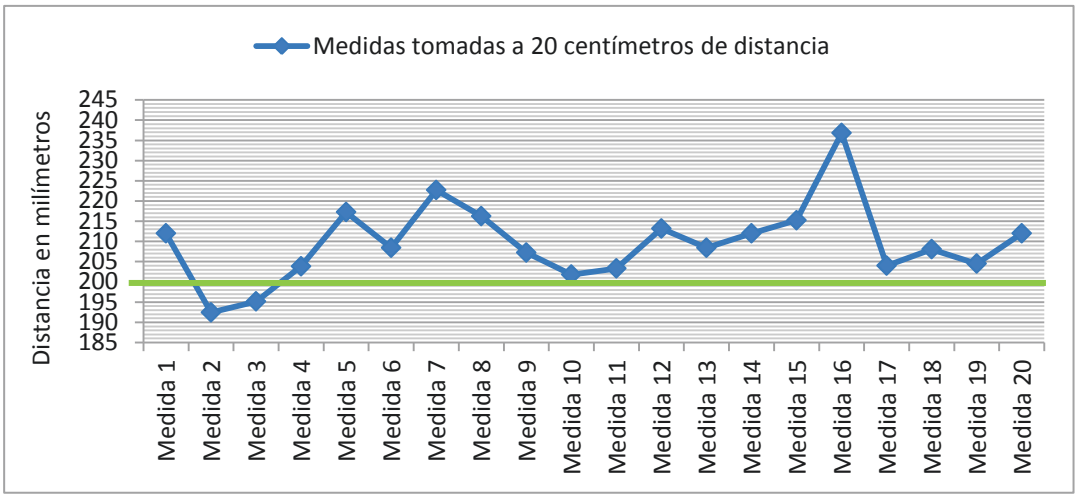
Las siguientes gráficas muestran, como las demás pruebas que se han realizado, un conjunto de muestras de medición consecutivas hasta un punto, tomadas mediante Arduino UNO y el sensor ultrasónico HC-SR04 a partir del *sketch* desarrollado. Esta vez, se ha probado a medir la distancia desde el sensor hasta la cara de una persona. La medida tomada como referencia es aproximada y se ha tomado hasta la frente de un sujeto situado de pie frente al sensor.

Prueba 5.1. Desde el sensor hasta una cara, a 15[cm] de distancia.



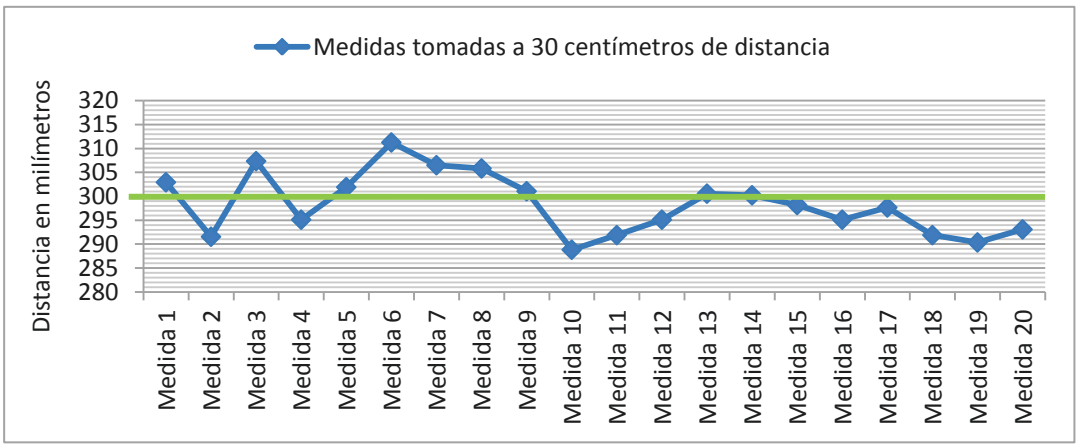
Prueba 5. 1

Prueba 5.2. Desde el sensor hasta una cara, a 20[cm] de distancia.



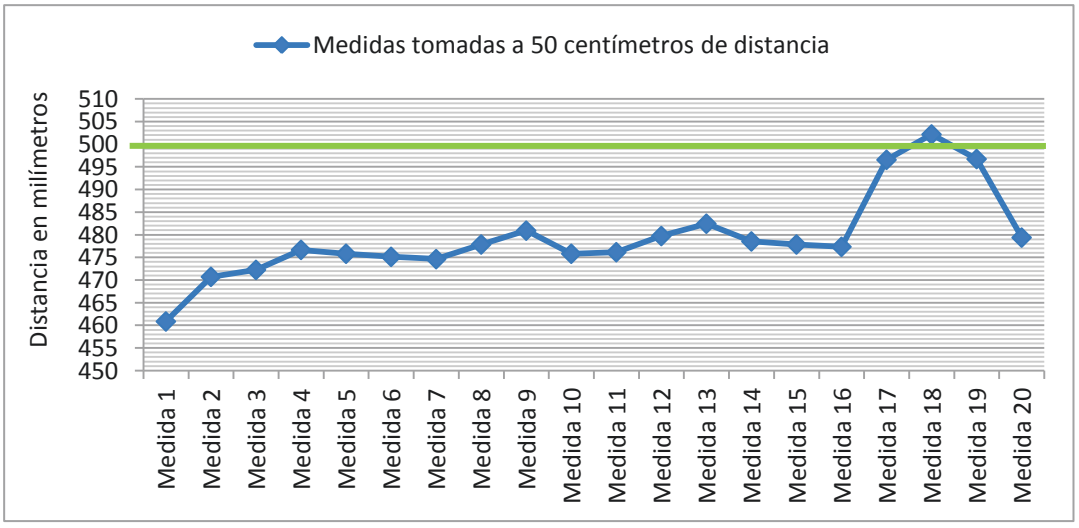
Prueba 5. 2

Prueba 5.3. Desde el sensor hasta una cara, a 30[cm] de distancia.



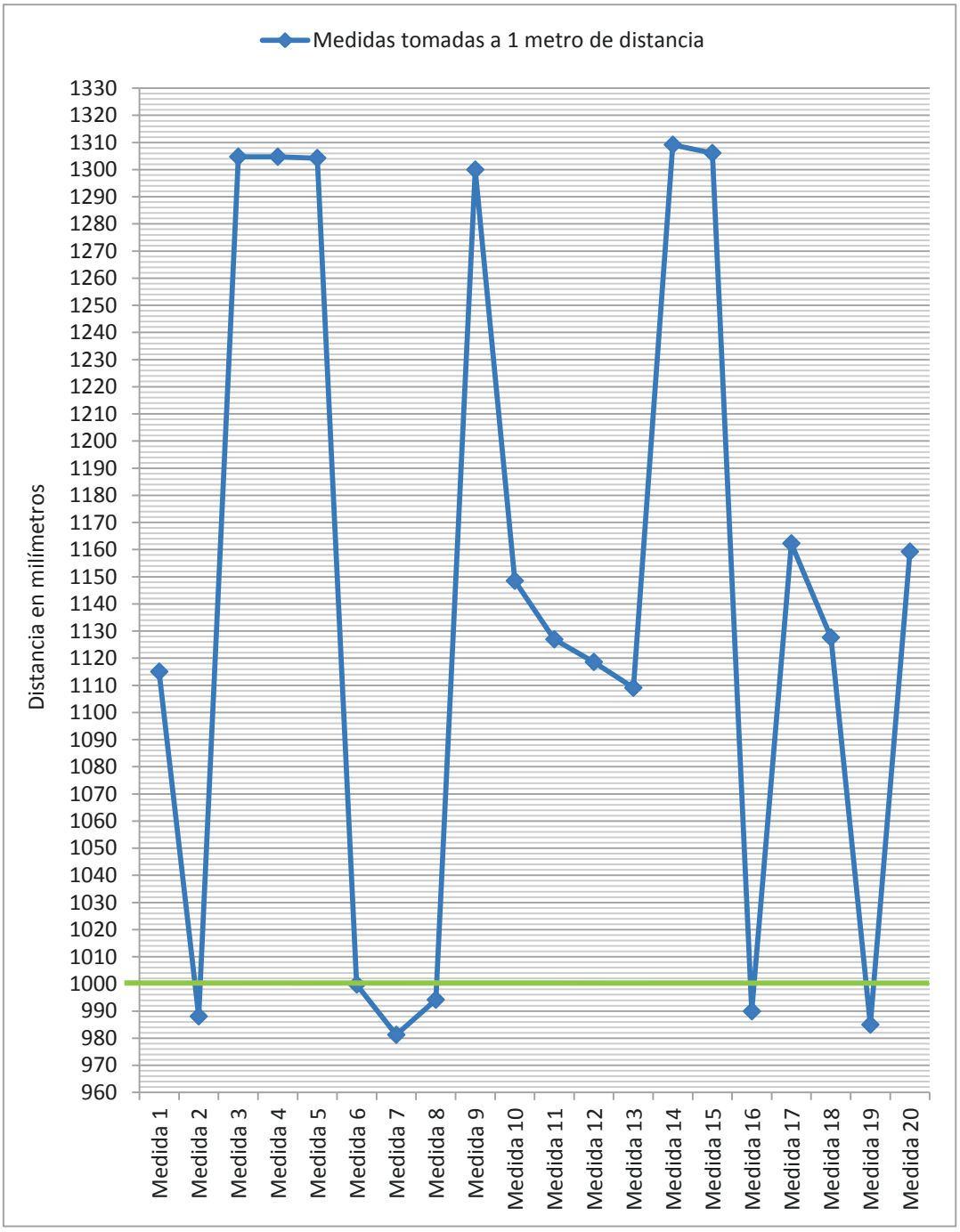
Prueba 5. 3

Prueba 5.4. Desde el sensor hasta una cara, a 50[cm] de distancia.



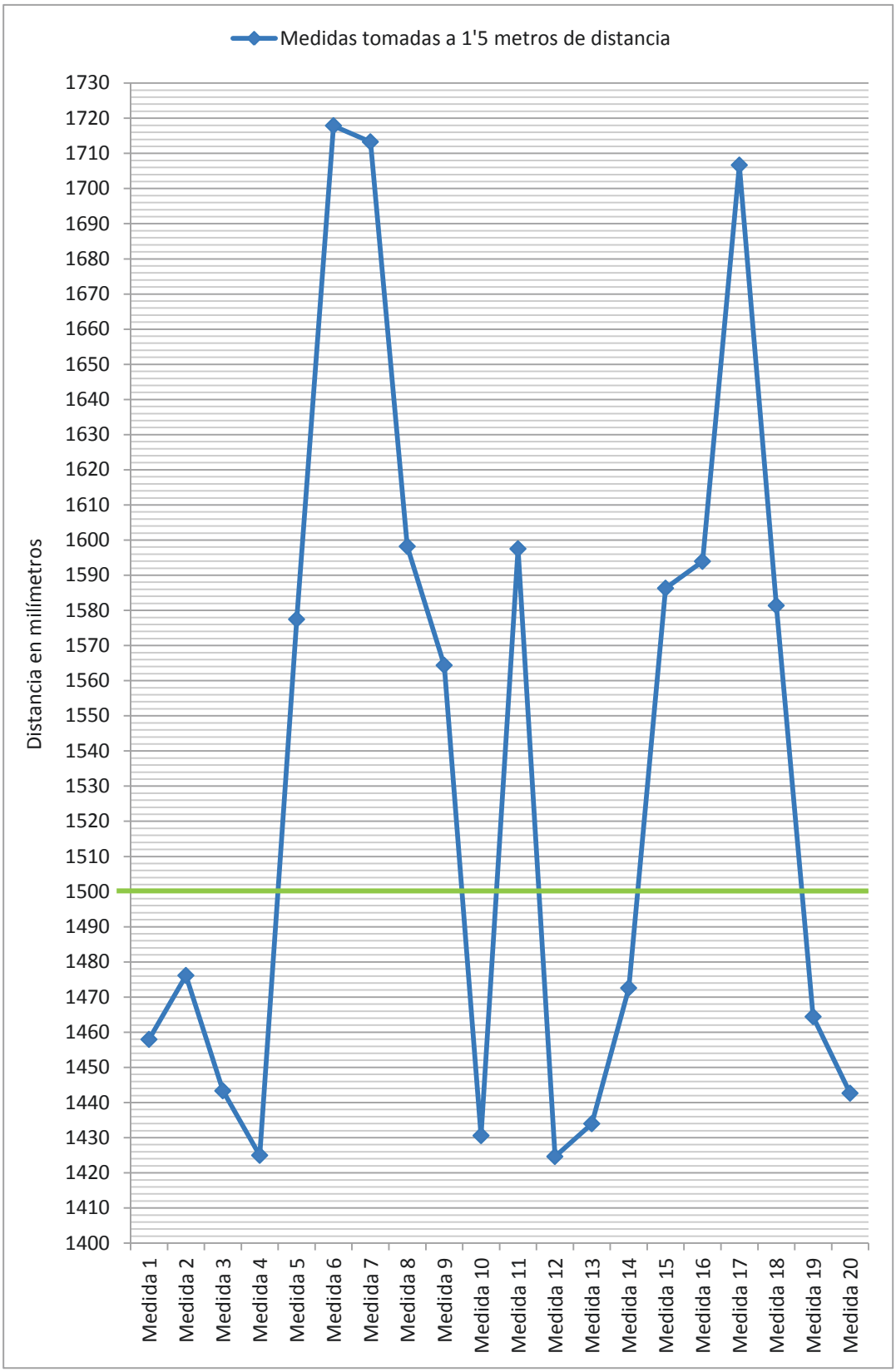
Prueba 5. 4

Prueba 5.5. Desde el sensor hasta una cara, a 1[m] de distancia.



Prueba 5. 5

Prueba 5.6. Desde el sensor hasta una cara, a 1'5[m] de distancia.



Prueba 5. 6

Prueba 5.7. Desde el sensor hasta una cara, a 2[m] de distancia.

A esta distancia se empiezan a apreciar picos de error. De la misma forma que ocurría en los casos anteriores en los que se ha experimentado con la medición de distancias a una superficie limitada en área, al tomar las medidas de distancia hasta la cara de una persona, las ondas de ultrasonidos dejan de rebotar únicamente en la cara de la persona y se desvían provocando mediciones que distan de la distancia real del sensor a la cara de la persona.

En la tabla que se muestran aparecen los resultados derivados de este experimento.

Teóricamente, en un espacio en el no hubiese otros objetos dentro de un ángulo de 15° desde el sensor hasta el objeto a medir (para evitar que las ondas ultrasónicas se desviasen), el sensor tendría una capacidad de medición fiable aproximadamente hasta 4[m] de distancia. Esto es difícil de conseguir con la cara de una persona, ya que no es una superficie uniformemente plana.

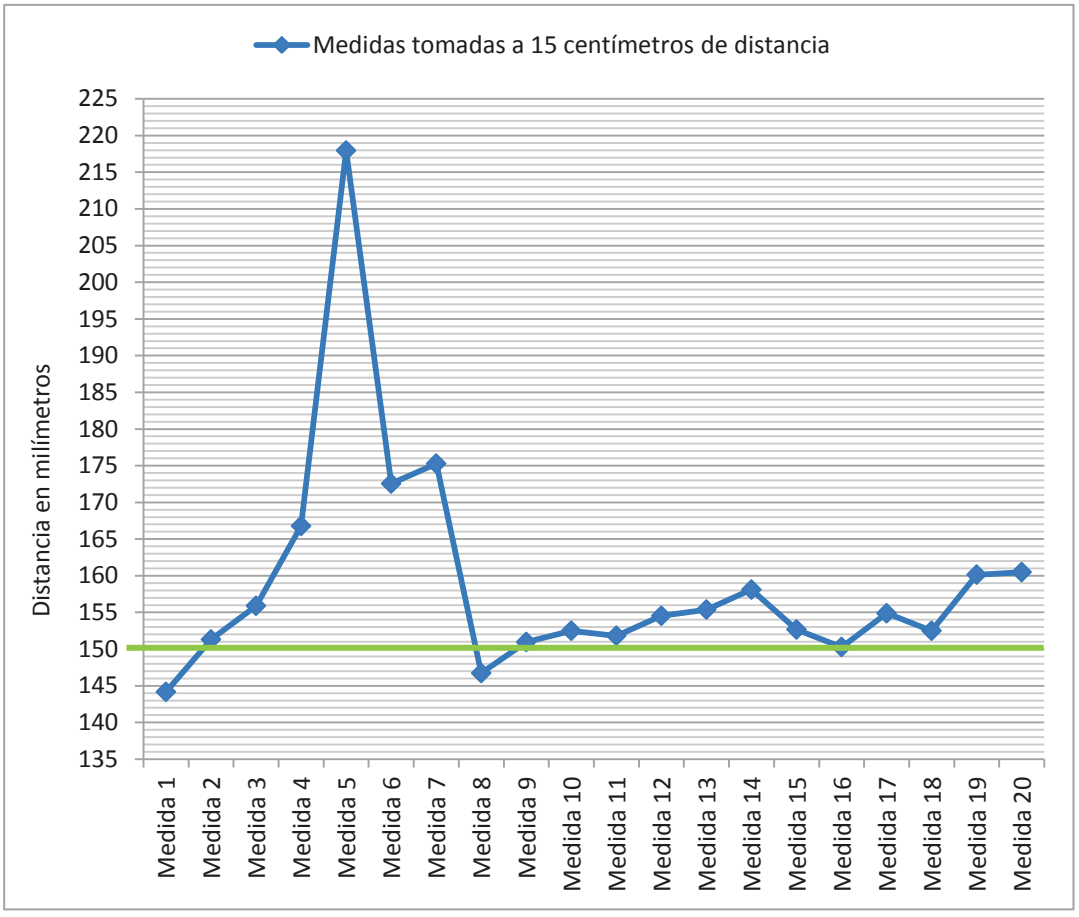
	Medidas tomadas a 2 [m]
Medida 1	2044.08mm
Medida 2	2039.83mm
Medida 3	2045.95mm
Medida 4	2024.36mm
Medida 5	71.23mm
Medida 6	51.34mm
Medida 7	2045.10mm
Medida 8	71.57mm
Medida 9	4674.32mm
Medida 10	71.57mm
Medida 11	71.57mm
Medida 12	50.49mm
Medida 13	71.57mm
Medida 14	2040.51mm
Medida 15	72.42mm
Medida 16	71.57mm
Medida 17	72.59mm
Medida 18	2045.27mm
Medida 19	2070.26mm
Medida 20	71.57mm

Prueba 5. 7

Prueba 6. Distancia a la mano de una persona.

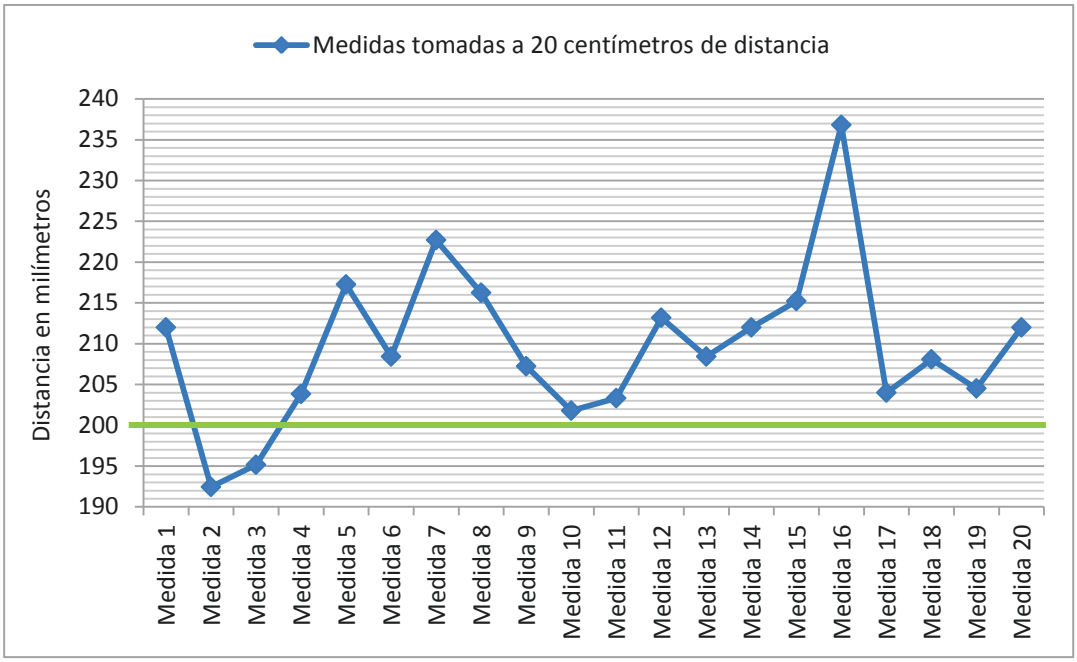
Para el desarrollo de estas pruebas, cuyo resultado se muestra en las siguientes gráficas, se ha situado la palma de la mano (en disposición vertical, con los dedos hacia arriba) de una persona frente al sensor HC-SR04. Puesto que se trata de una extremidad del cuerpo humano tiene un movimiento inherentemente asociado a él y no puede permanecer constantemente a una distancia exacta del sensor. La distancia que se toma como referencia (al igual que en el resto de pruebas), a partir de un instrumento de medida, como una regla o un metro (en las distancias más cortas), o el sensor láser (en distancias mayores, a partir de 20[cm)], será siempre, una distancia de referencia aproximada.

Prueba 6.1. Desde el sensor hasta una mano a 15[cm] de distancia.



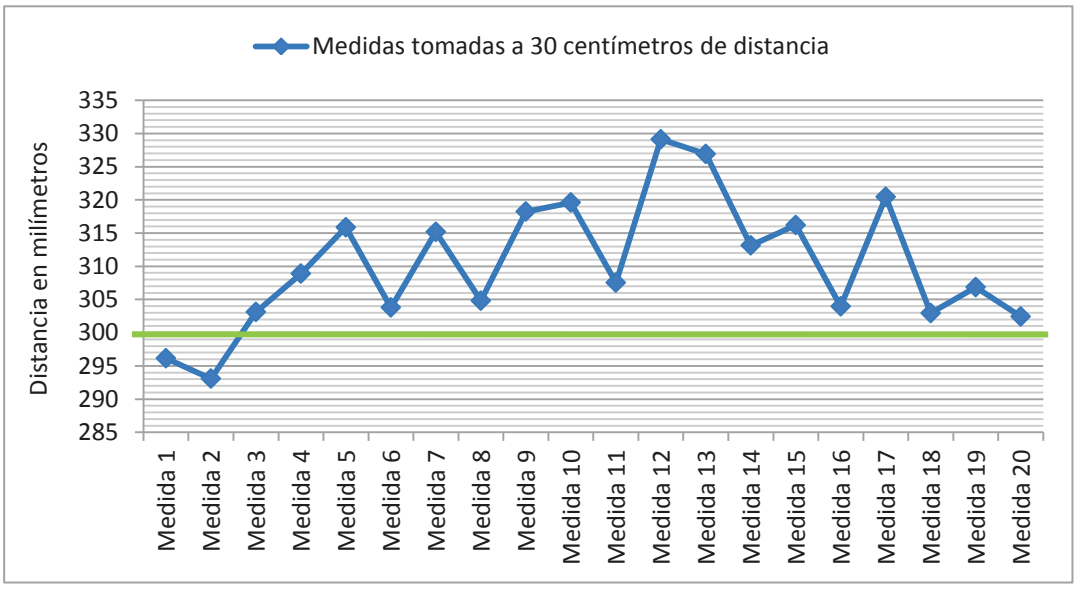
Prueba 6. 1

Prueba 6.2. Desde el sensor hasta una mano a 20[cm] de distancia.



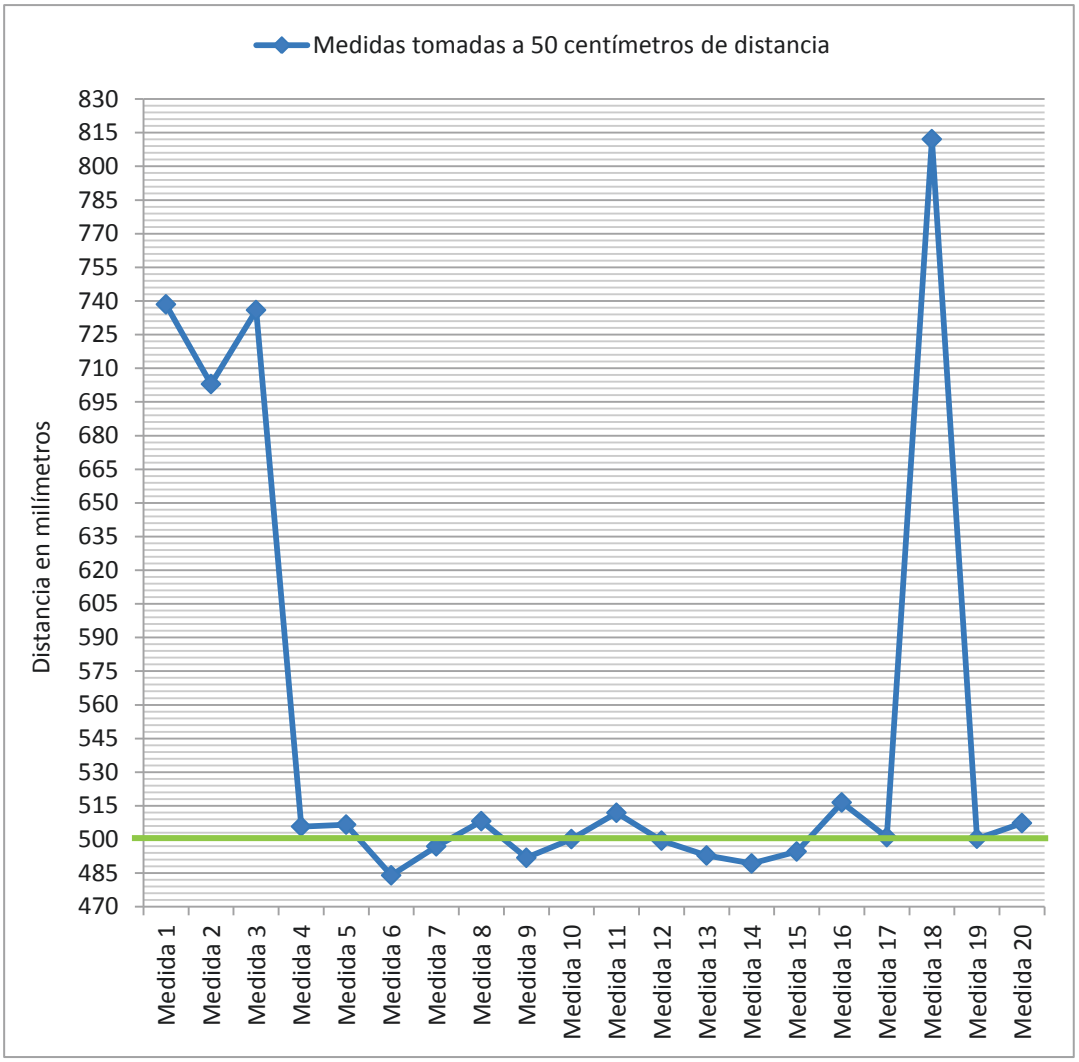
Prueba 6. 2

Prueba 6.3. Desde el sensor hasta una mano a 30[cm] de distancia.



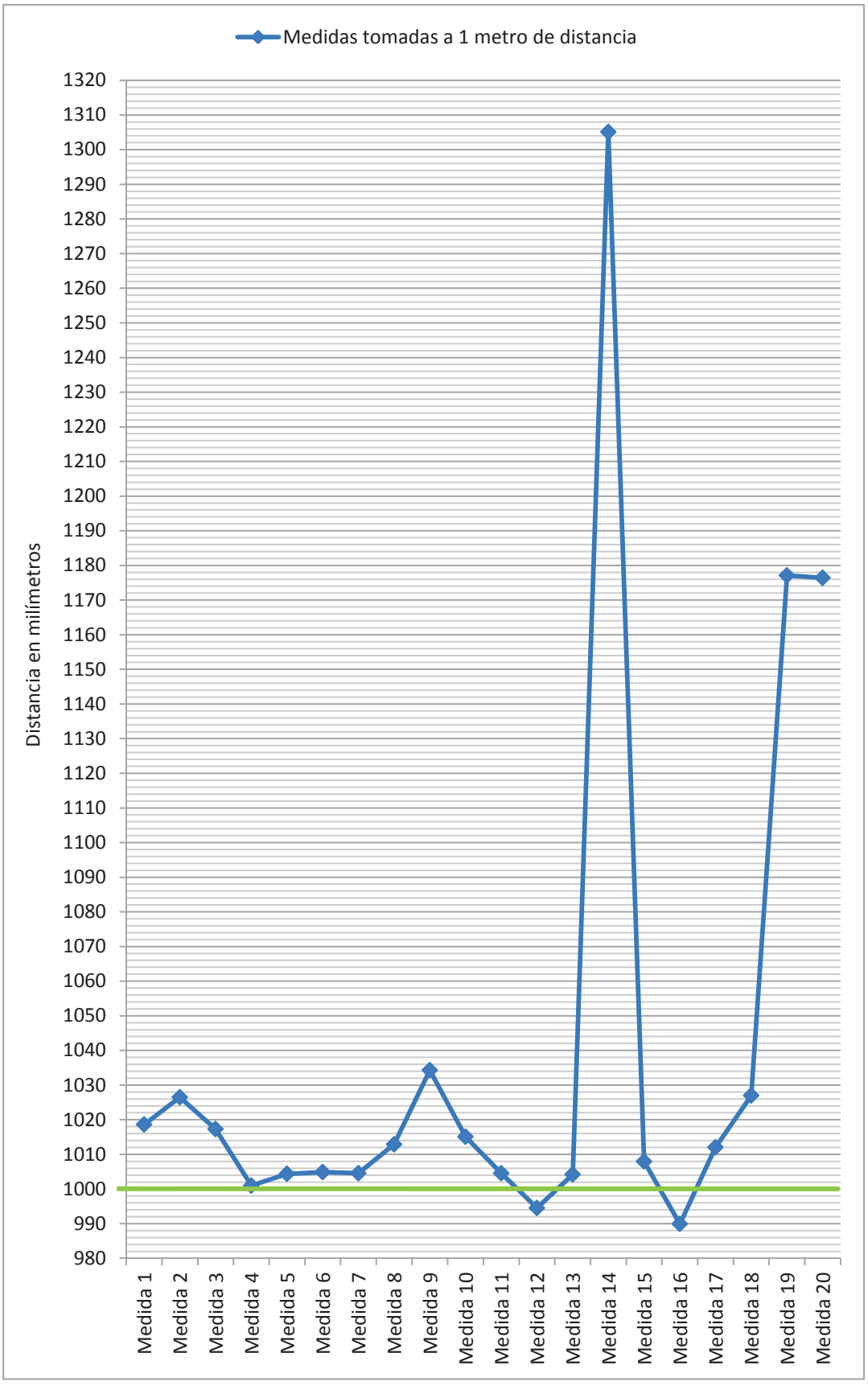
Prueba 6. 3

Prueba 6.4. Desde el sensor hasta una mano a 50 [cm] de distancia.



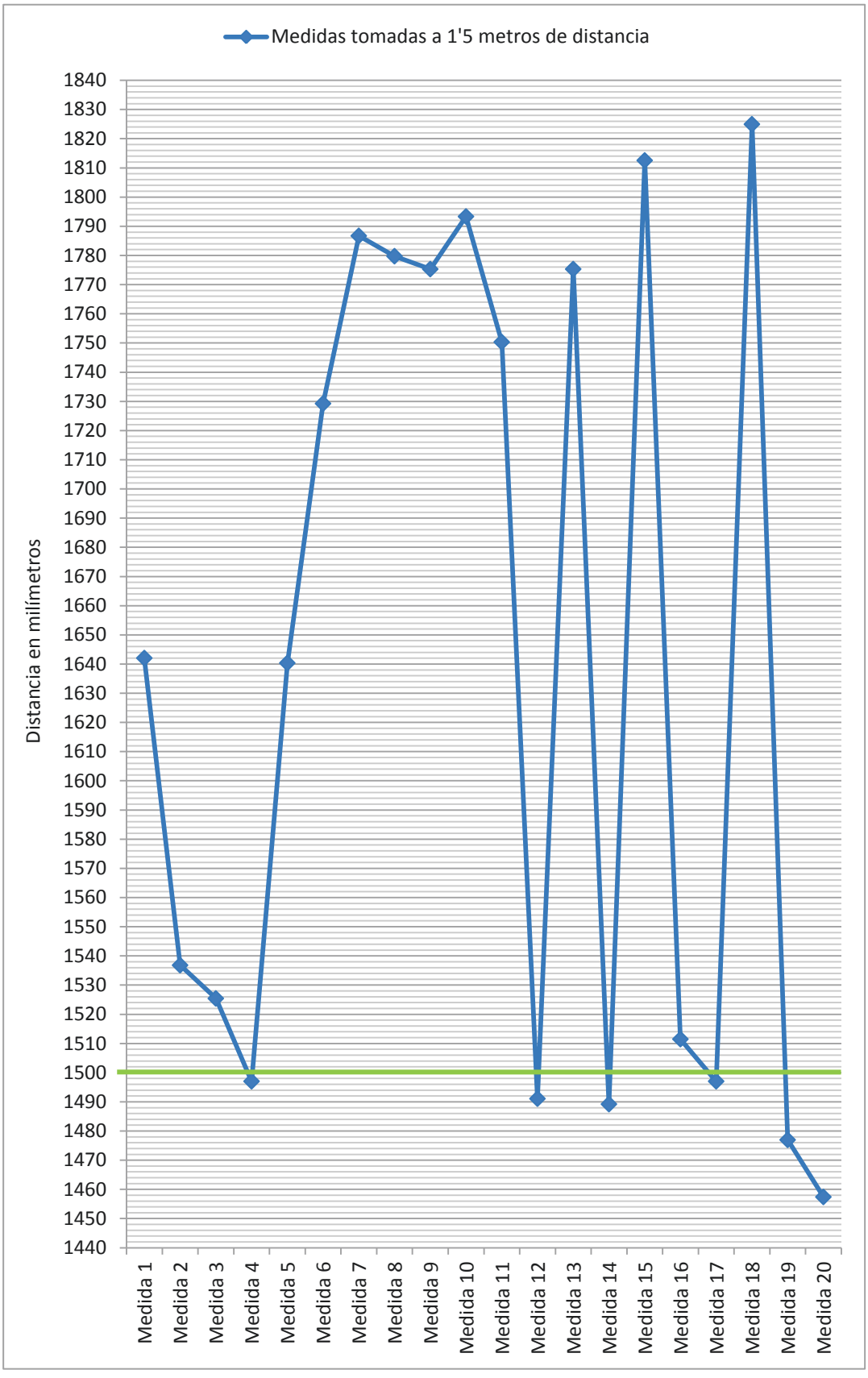
Prueba 6. 4

Prueba 6.5. Desde el sensor hasta una mano, a 1[m] de distancia.



Prueba 6. 5

Prueba 6.6. Desde el sensor hasta una mano, a 1'5[m] de distancia.



Prueba 6. 6

Prueba 6.7. Desde el sensor hasta una mano, a 2[m] de distancia.

A esta distancia se empiezan a apreciar picos de error. De la misma forma que ocurría en los casos anteriores en los que se ha experimentado con la medición de distancias a una superficie limitada en área, al tomar las medidas de distancia hasta la mano de una persona, las ondas de ultrasonidos dejan de rebotar únicamente en la mano y se desvían, provocando algunas mediciones erróneas.

En la tabla que se muestra a la izquierda aparecen los las muestras de medición tomadas en este experimento.

Teóricamente, en un espacio en el no hubiese otros objetos dentro de un ángulo de 15° desde el sensor hasta el objeto a medir (para evitar que las ondas ultrasónicas se desviasen), el sensor tendría una capacidad de medición fiable aproximadamente hasta 4[m] de distancia. Lo cual es difícil de llevar a la práctica con la mano de una persona, puesto que no es una superficie uniforme.

	Medidas tomadas a 2[m]
Medida 1	1955.34mm
Medida 2	1946.16mm
Medida 3	1993.93mm
Medida 4	2003.96mm
Medida 5	1982.71mm
Medida 6	2069.41mm
Medida 7	2039.49mm
Medida 8	51.34mm
Medida 9	2396.83mm
Medida 10	2045.27mm
Medida 11	2035.41mm
Medida 12	2404.31mm
Medida 13	2086.75mm
Medida 14	2027.25mm
Medida 15	2023.00mm
Medida 16	2035.92mm
Medida 17	72.42mm
Medida 18	1971.15mm
Medida 19	2027.25mm
Medida 20	2036.09mm

Prueba 6. 7

Conclusiones

Reflexiones

A partir de este trabajo de investigación sobre Arduino UNO junto al sensor ultrasónico HC-SR04 en cuanto a medición de distancias se refiere, se pueden llevar a cabo algunas reflexiones.

Por qué utilizar Arduino y no otra tecnología similar

El sistema Arduino se diferencia de otras plataformas existentes en el mercado (tal y como aclara Massimo Banzi, cofundador de Arduino), esencialmente en que: es un entorno de plataforma múltiple (**multiplataforma**) que se puede ejecutar en los **principales sistemas operativos** Windows, Mac OS X y Linux; su IDE está fundamentado en el lenguaje de programación Processing, un entorno de desarrollo fácil de usar, utilizado sobre todo por artistas y diseñadores; se **programa fácilmente mediante un cable con conector USB**, y no a través de un puerto serie en sí (del cual no disponen algunos equipos modernos); su *software* y *hardware* es **libre** (se pueden descargar su IDE y sus diagramas del circuito, comprar los componentes y crear una placa propia, sin tener que pagar nada por ello a los creadores de Arduino); el *hardware* es **económico** (una placa Arduino USB completa cuesta unos 30€ y además, sus chips se pueden reemplazar, por lo que permite cometer fallos); es una **comunidad** de usuarios activa y en constante crecimiento, por lo que hay muchas personas que pueden ayudar; se desarrolló en un entorno **educativo**, así que se puede conseguir que proyectos sencillos funcionen rápidamente sin ser un experto en informática o electrónica.

En la actualidad existen diversas plataformas además de Arduino para, aplicando la ingeniería informática, desarrollar productos electrónicos a bajo coste. Dos variantes de las placas Arduino, que se pueden programar mediante el mismo IDE que Arduino pueden ser la tarjeta Nanode[95] (una placa electrónica de código abierto conectada a Internet) o la placa Libelium Waspmote[96] (una plataforma modular, también *open-source,* diseñada para construcción de redes de sensores inalámbricas de bajo consumo). Otra plataforma similar a Arduino, puede ser Beaglebone[97], con mayores ventajas a

[95] Si se desea profundizar más en Nanode se puede ir a: http://www.nanode.eu/classic-build/.
[96] Para conocer Libelium Waspmote se puede acceder a: http://www.libelium.com/es/products/waspmote/.
[97] Para mayor información sobre Beaglebone se puede consultar en el enlace: http://beagleboard.org/bone.

la hora de crear **proyectos con sensores externos o redes**. Pero **las más extendidas** y con el mayor número de usuarios actualmente son las tarjetas **Raspberry Pi**[98] **y las placas Arduino**.

Arduino o Raspberry pi

Su principal deferencia es que la placa Arduino se basa en un **microcontrolador**, mientras que la placa Raspberry Pi se basa en un **microprocesador** (de 256 ó 512[MB] de memoria RAM). Arduino trabaja a una frecuencia **16[MHz]** y Raspberry Pi lo hace a **700[MHz]**. Además, Arduino no consta de **sistema operativo** propio, mientras que Raspberry Pi incorpora uno propio, de GNU/Linux (el más común es Raspbian[99], un sistema operativo libre basado en Debian y optimizado para el *hardware* Raspberry Pi). Por otra parte, Arduino, para conectarse a Internet, necesita un adaptador de Ethernet, mientras que Raspberry Pi incluye una salida para Ethernet, varias USB y HDMI.

Ambas fueron originalmente diseñadas con fines docentes. Arduino, por su parte, es una tarjeta más económica y sencilla de programar, interesante para principiantes y **proyectos de un destino específico**. Por ello, Raspberri Pi se suele reservar para **proyectos de** complejidad más elevada, que exijan **mayor potencia de cálculo y procesamiento**, trabajos multimedia, o proyectos basados en Linux.

Se ha de **elegir una plataforma u otra en función del uso concreto** que se vaya hacer de la misma. La simplicidad de **Arduino** permite que sea una buena elección a la hora de desarrollar **proyectos orientados al *hardware***. Mientras que **Raspberry Pi** es **más potente en relación al *software*.** Puesto que este proyecto se fundamenta en la interacción con el *hardware* a través de la implementación de un sencillo programa, la mejor elección para desarrollarlo, ha sido servirse de Arduino.

[98] El sitio web oficial de Raspberry Pi es: http://www.raspberrypi.org/.
[99] Para más información se puede consultar el sitio web de Raspbian: http://www.raspbian.org/.

Líneas Futuras

Incremento de la precisión de la aplicación desarrollada

A la velocidad del sonido, y por lo tanto también a la **de los ultrasonidos**, **le afectan** valores como **la temperatura, la humedad y la latitud**. En este proyecto, para el desarrollo de una sencilla aplicación para la medición de distancias mediante Arduino UNO y el sensor HC-SR04, se ha tomado la velocidad constante del sonido de 340[°C], suponiendo una temperatura de 25[°C], una humedad de[] y una latitud de[], puesto que la idea principal del trabajo de investigación se ha centrado únicamente en la interacción de Arduino con un único sensor. Pero en desarrollo más complejos, donde primase el menor grado de error posible y la interacción de Arduino con varios sensores y la de sensores entre sí, añadiendo un sensor de temperatura al circuito, se podría aumentar la precisión del mismo, de forma que, la velocidad del ultrasonido se viese influenciada según la temperatura del medio en que se encontrase la placa Arduino UNO junto al sensor HC-SR04.

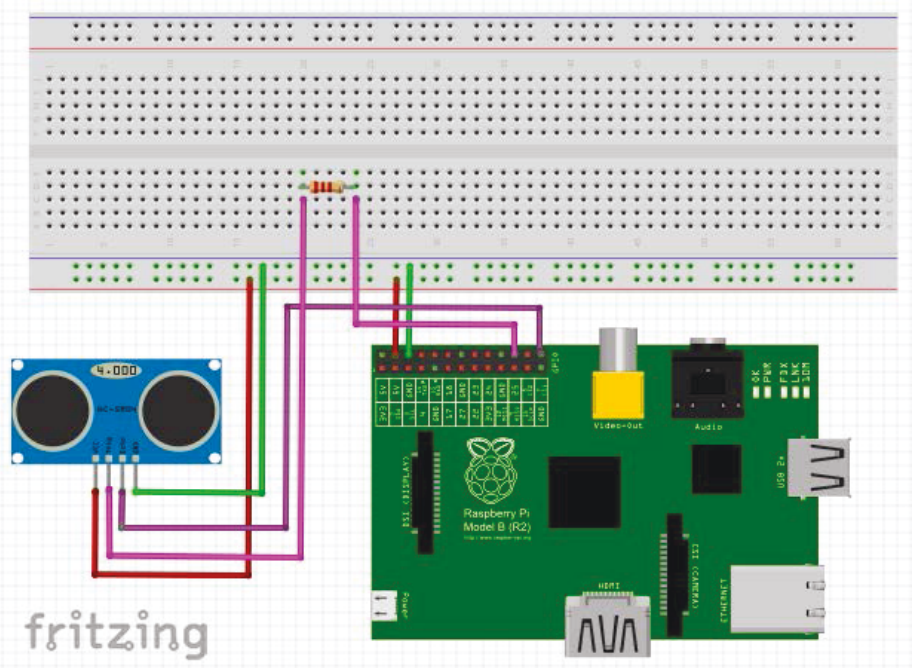
Medición de distancias con Raspberry Pi B y HC-SR04

Para el desarrollo de la aplicación de medición de distancias por ultrasonidos con el sensor HC-SR04, en este proyecto se ha utilizado la plataforma Arduino y en concreto la placa Arduino UNO. Pero también podría haberse utilizado otra plataforma similar, como es Raspberry-Pi.

El diseño de un **circuito** para medición de distancias por ultrasonidos mediante el sensor HC-SR04 con **Raspberry Pi**, sería similar al diseño del circuito que se ha diseñado empleando Arduino UNO. La patilla Vcc (para alimentación) del HC-SR04 se conectaría al pin GPIO de 5V de la Raspberry Pi, la patilla Gnd (para toma de tierra) del HC-SR04 se conectaría al pin GPIO de la Raspberry Pi, la patilla Trig del HC-SR04 se conectaría a un pin GPIO de la Raspberry Pi, y la patilla Echo del HC-SR04 se conectaría a otro pin GPIO de la Raspberry Pi. Pero **la señal de salida del sensor** (a través de su patilla Echo) **es de 5[V],** mientras que **los pines** GPIO **de la Raspberry Pi** tienen una tensión de **funcionamiento de 3'3[V],** así que si se le suministran 5[V] a un pin de la Raspberry Pi, podría dañarse. Es necesario, por lo tanto, reducir el voltaje de salida (patilla Echo) del HC-SR04, al voltaje de trabajo de la Raspberry Pi, 3'3[V]. Para ello, se puede optar bien por añadir una resistencia de 1**[KΩ]** (entre la patilla Echo del HC-SR04 y un el pin GPIO de la Raspberry Pi al que vaya a conectarse) o bien, utilizar un **divisor de tensión**, que consta de **dos resistencias (una resistencia de 1[KΩ] y una**

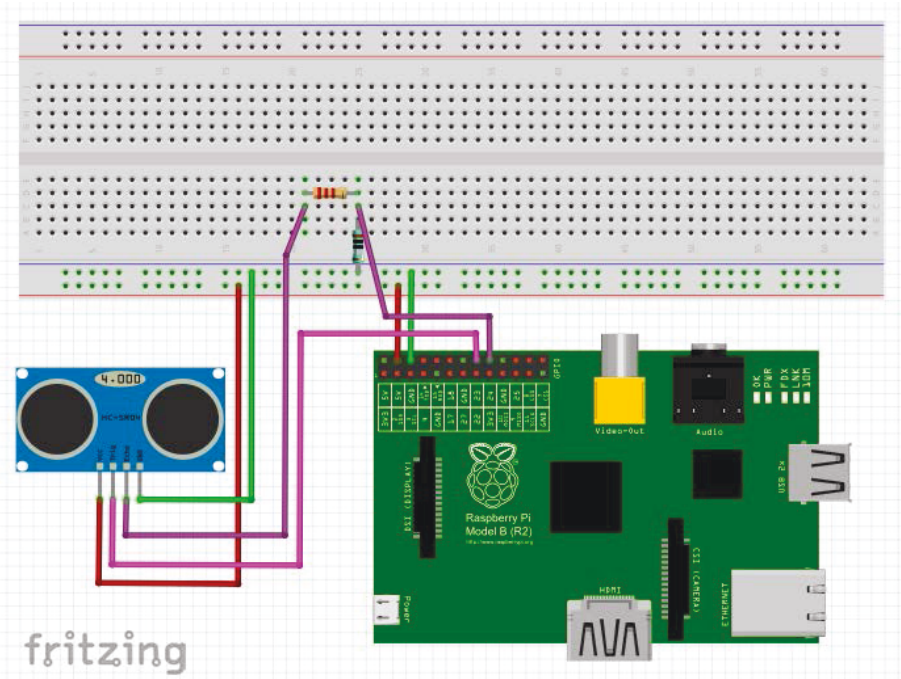
resistencia de 2[KΩ]), conectadas en serie (entre la patilla Echo del HC-SR04 y el pin GPIO de la Raspberry-Pi al que vaya a conectarse).

Un posible diseño de este circuito utilizando una resistencia de 1[KΩ] es el que se muestra en esta imagen.



Líneas futuras, imagen 1

Un posible diseño del circuito, utilizando un divisor de tensión (formado por una resistencia de 1[KΩ] y una resistencia de 2[KΩ]) es el que se muestra en esta imagen.



Líneas futuras, imagen 2

La implementación del programa, una vez diseñado el cableado del circuito, puede realizarse en un *script* mediante el lenguaje de programación Python. La secuencia de comandos en Python (como ocurría en Arduino) debería recoger la duración del pulso (en base al tiempo de ida y vuelta de la ráfaga de ultrasonidos desde y hasta al sensor) para calcular la distancia entre el HC-SR04 y el objeto que se ponga delante de su ángulo de actuación.

Anexos

Anexo I: El lenguaje de programación Arduino

Los elementos que conforman los programas de Arduino se pueden dividir principalmente en tres partes: estructuras, valores (variables y constantes), y funciones[100].

Estructuras

Bloques

- `setup()`. Esta función se invoca al iniciarse el *sketch* y se ejecuta una sola vez, después de cada arranque o *reset* de la placa Arduino. Inicia y establece los valores iniciales. Se suele utilizar para iniciar las variables y para configurar el modo de los *pines*.
- `loop()`. Permite repetir continuamente, en forma de bucle, el código que incorpora. Es ejecutada a continuación del `setup()`. Se utiliza para gestionar el control de la placa Arduino, leyendo las entradas, procesando los datos, escribiendo en las salidas, etc.

Estructuras de control

- `if`, `if…else`, `for`, `switch case`, `while`, `do… while`, `break`, `continue`, `return`, y `goto`.

Elementos sintácticos

- "`;`", "`{}`", "`//`", "`/* */`", `#define`, e `#include`.

Operadores aritméticos

- `=`, `+`, `-`, `*`, `/`, y `%`.

Operadores de comparación

- `==`, `!=`, `<`, `>`, `<=` y `>=`.

[100] Cada una de las referencias al lenguaje de programación Arduino se pueden consultar con más detalle en: http://arduino.cc/en/Reference/HomePage.

Operadores booleanos

- `&&` (and), `||` (or) y `!` (not).

Operadores de bits

- `&`, `|`, `^`, `~`, `<<` y `>>`.

Operadores compuestos

- `++`, `--`, `+=`, `*=`, `/=`, `&=` y `|=`.

Indicadores de acceso a operadores

- `*` y `&`.

Valores: variables y constantes

Constantes

- `HIGH/LOW`, `INPUT/OUTPUT/INPUT_PULLUP`, `true/false`, `integer constants` y `floating point constants`.

Tipos de datos

- `void`, `boolean`, `char`, `unsigned char`, `byte`, `int`, `unsigned int`, `word`, `long`, `unsigned long`, `short`, `float`, `double`, `string-char array`, `String-object array`.

Conversión de tipos

- `char()`, `byte()`, `int()`, `word()`, `long()` y `float()`.

Clasificación de variables

- `static`, `volatile` y `const`.

Utilidades

- `sizeof()`.

Funciones

Entradas/salidas digitales

- `pinMode()`, `digitalWrite()`, `digitalRead()`.

Entradas/salidas analógicas:

- `analogReference()`, `analogRead()`, analogWrite()-PWM.
- `analogReadResolution()`, `analogWriteResolution()`.

Entradas/salidas avanzadas

- `tone()`, `noTone()`, `shiftOut()`, `shiftIn()`, `pulseIn()`.

De tiempo

- `millis()`, `micros()`, `delay()`, `delayMicroseconds()`.

Matemáticas

- `min()`, `max()`, `abs()`, `constrain()`, `map()`, `pow()`, `sqrt()`.

Trigonométricas

- `sin()`, `cos()`, `tan()`.

Para números aleatorios

- `randomSeed()`, `random()`.

Bits y Bytes

- `lowByte()`, `highByte()`, `bitRead()`, `bitWrite()`, `bitSet()`, `bitClear()` y `bit()`.

Interrupciones externas

- `attachinterrupt()` y `detachinterrupt()`.

Interrupciones

- `interrupts()` y `noInterupts()`.

De comunicación

- `Serial` y `Stream`.

Para USB (solo en placasLeonardo y Due)

- `Keyboard()` y `Mouse()`.

Anexo II: Los distintos tipos de sensores

En ocasiones, es interesante medir y/o controlar una o varias magnitudes físicas de forma automática, sin la intervención humana. Para ello, se necesita un elemento sensible a dicha magnitud física que nos permita evaluar su variación; es decir, lo que conocemos comúnmente como sensor. Existen sensores de muchos tipos. A continuación se muestran sus clasificaciones más comunes.

Según su funcionamiento o necesidad de energía externa:

- **Activos**: necesitan una fuente de energía externa, normalmente, un circuito de polarización. La señal externa aplicada es modulada en el sensor, por la magnitud física a medir, dando lugar a la señal de salida.
  Ejemplos: Sensores resistivos, magnetoresistores, fototransistores, etc. Sensores adecuados para medir señales débiles.
- **Pasivos**: se bastan de las propias condiciones medioambientales, sin necesidad de una fuente de energía externa. Son capaces de generar una señal eléctrica sin necesidad de una fuente de energía (eléctrica) auxiliar.
  Ejemplos: Termoacoplador, y muchos piezoeléctricos.

Según las señales que proveen:

- **Analógicos**: proporcionan información a través de una señal analógica (como tensión o corriente), pudiendo tomar infinitos valores entre un mínimo y un máximo.
  Ejemplos: potenciómetros
- **Digitales**: dan información mediante una señal digital (0 ó 1 lógicos, o bits).
  Ejemplos: codificadores de posición.

Según la naturaleza de su funcionamiento:

- **De posición**: experimentan variaciones según la posición que ocupan en cada instante los elementos que los componen.
- **Fotoeléctricos**: según la luz que incide sobre ellos.
- **Magnéticos**: en función del campo magnético que les atraviesa.
- **De temperatura**: en función de la temperatura del lugar donde se encuentran.
- **De humedad**: según el nivel de humedad del medio en que se encuentran.
- **De presión**: según la presión a la que son sometidos.

- **De movimiento**: en función de los movimientos a los que son sometidos.
- **Químicos**: en función de los agentes químicos externos que pueden incidir sobre ellos.

Según los elementos que se utilizan en su fabricación:

- **Mecánicos**: utilizan contactos mecánicos que se abren o cierran.
- **Resistivos**: utilizan elementos resistivos.

  Ejemplos: Galga.
- **Capacitivos**: utilizan condensadores.

  Ejemplos: Dieléctrico variable.
- **Inductivos**: utilizan bobinas.
- **Piezoeléctricos**: utilizan cristales como el cuarzo.
- **Semiconductores**: utilizan semiconductores.

Según la magnitud (física o química) que miden:

- De medición de magnitudes **térmicas**: normalmente miden magnitudes como la temperatura y el flujo de calor.

  Ejemplos:
- De medición de magnitudes **químicas**: miden las propiedades internas de la materia, como la concentración de un cierto material, la composición o la velocidad de reacción.
- De medición de magnitudes **magnéticas**: las más comunes son la intensidad o dirección de campo magnético, la densidad de flujo y la magnetización.
- De medición de magnitudes **de radiación electromagnética**, como pueden ser la intensidad, la longitud de onda, la polarización y la fase.
- De medición de magnitudes **eléctrica**s: fundamentalmente magnitudes como la tensión, la corriente y la carga.
- De medición de magnitudes **mecánicas**: miden magnitudes mecánicas como pueden ser la fuerza (vectorial), la presión (escalar), la velocidad (vectorial), la aceleración (vectorial) y la posición (lineal o angular).

  Ejemplos: Encoders y Resolvers.
- De medición de magnitudes **escalares**: como la distancia.

  Ejemplos: Infrarrojos y **Ultrasonidos**.

Anexo III: Los sistemas electrónicos y su producción

Un **sistema electrónico** es está formado por sensores, por la circuitería de procesamiento y control, por actuadores o activadores, y por una fuente de alimentación. Los sensores, obtienen información del mundo físico y la transforman en una señal eléctrica que pueda ser manejada por la circuitería interna de control. La **circuitería interna de procesamiento y control** en sí, procesa la señal eléctrica según el diseño de los componentes *hardware* del sistema y el *software* (conjunto lógico de instrucciones o programa) que dicho *hardware* tenga pregrabado y ejecute de forma autónoma. Los **activadores o actuadores** (*actuators*) transforman la señal procesada por la circuitería interna, en energía que actúa directamente sobre el medio. La **fuente de alimentación** proporciona la energía necesaria para que se pueda realizar todo el proceso.

Un **sistema integrado** es un sistema electrónico, formado por *hardware* y *software*. Su parte central normalmente es el *hardware* **procesador** (en forma de microprocesador o microcontrolador), a partir del cual será necesario un ***software*** y un **compilador** capaz de programarlo.

A lo largo del **proceso de producción electrónica** hay una serie de etapas que, aunque en pueden variar en función de la complejidad de cada proyecto, en general, se desarrollan desde la idea de un producto, hasta su materialización en el producto final. En primer lugar, surge una **idea** para mejorar algo o solucionar un problema, se extrae la esencia de la idea y se modela la solución más óptima. A continuación, se diseña el **prototipo** (mediante Arduino, a ello se ha orientado este proyecto) para implementar la solución, que satisfaga los intereses a los que se pretendía llegar y con los elementos correctos para su montaje. Cuando se comprueba que el circuito del prototipo funciona correctamente, se crean los **fotolitos** (planos del circuito en una placa con las conexiones y elementos necesarios para su producción comercial) y la propia **tarjeta de circuito impreso** (una PCB, *Printed Circuit Board*, o superficie no conductora con pistas conductoras normalmente de cobre trazadas en ella, y a las cuales se fijan los componentes del circuito completo. Finamente, se realiza la **unión de los componentes a la tarjeta**, soldándolos (habitualmente con estaño o plomo aunque, cada vez más, se están manejando otras alternativas) para hacer posible la unión mecánica de los componentes y la unión eléctrica, que mantiene la conductividad entre los componentes y las pistas.

.

Bibliografía

Las referencias bibliográficas consultadas para la elaboración de este trabajo de fin de grado, se muestran a continuación, atendiendo al estándar que indica la Norma ISO 690:2010 y su versión española UNE-ISO 690.

Libros consultados:

Ribas Lequerica, Joan. ***Arduino práctico, manual imprescindible***. [En papel]. Madrid: Ediciones Anaya Multimedia (Grupo Anaya S.A.), 2014. 400 p. ISBN: 978-84-415-3419-3.

Torrente Artero, Óscar. ***Arduino, curso práctico de formación***. [En papel]. Madrid: RC Libros, 2013. 588 p. ISBN: 978-84-940725-0-5.

Banzi, Massimo. ***Introducción a Arduino***. [En papel]. Madrid: Ediciones Anaya Multimedia (Grupo Anaya S.A.), 2012. 128 p. ISBN: 978-84-415-3177-2.

Lajara Vizcaíno, José Rafael. Pelegrí Sebastià, José. ***Sistemas integrados con Arduino***. [En papel]. Barcelona: Marcombo S.A., 2014. 308 p. ISBN: 978-84-267-2093-1.

Serna Ruiz, Antonio. Ros García, Francisco Antonio. Rico Noguera, Juan Carlos. ***Guía práctica de sensores***. [En papel]. Madrid: Creaciones Copyrigth, S.L., 2010. 226 p. ISBN: 978-84-92779-49-9.

Morón Fernández, Carlos. García García, Alfonso. ***Sensores y actuadores***. [En papel] Madrid: Escuela Politécnica de Madrid, (Fundación general), 2009. 140 p. ISBN: 978-84-96737-53-2.

Sitios web de referencia:

Arduino. Arduino Website, 2014 [consulta 2014]. Disponible en: http://www.arduino.cc.

Fritzing. Fritzing, 2014 [consulta 2014]. Disponible en http://www.fritzing.org.

Raspberri Pi. Raspberry Pi, 2014 [consulta 2014]. Disponible en: http://www.raspberrypi.org/.

Además de los sitios web que se han nombrado en diversas notas al pie a lo largo de este documento, y multitud de videotutoriales en Internet.

FIN